

**PETRE PREOTEASA
LUCA-DAN ȘERBĂNAȚI**

**MATEMATICĂ
APLICATĂ
ÎN
TEHNICA
DE CALCUL**

XI



PETRE PREOTEASA
LUCA – DAN ȘERBANAȚI

MATEMATICĂ APLICATĂ ÎN TEHNICA DE CALCUL

XI

Manual pentru clasa a XI-a
licee de matematică-fizică



EDITURA DIDACTICĂ ȘI PEDAGOGICĂ
BUCHUREȘTI

Capitolul I

ELEMENTE DE BAZĂ ALE PROGRAMĂRII CALCULATOARELOR

Limbajele de programare sînt mijloace de comunicare ale omului cu calculatorul. Ele sînt inventate de om și servesc utilizatorilor pentru a exprima ceea ce trebuie să execute calculatorul.

Se știe că un calculator este un obiect inert atîta vreme cît nu i se indică operațiile pe care trebuie să le execute. Aceste operații sînt exprimate de către utilizator sub forma unor comenzi sau indicații și transmise astfel calculatorului.

Succesiunea de comenzi sau indicații transmise calculatorului în vederea executării lor se numește *program*. O comandă sau indicație dintr-un program se numește *instrucțiune*.

Exprimînd într-un program *ceea ce trebuie să facă* un calculator, utilizatorul îi transmite de fapt un algoritm. Limbajul de programare ne dă tocmai modul în care descriem acest algoritm. O definiție echivalentă a programului va fi deci următoarea :

Un program este descrierea unui algoritm într-un limbaj de programare.

Algoritmul, concept abstract, capătă prin program o reprezentare simbolică. După cum se știe, o altă modalitate de reprezentare (descriere) simbolică a unui algoritm este schema logică.

În primul capitol al manualului vom introduce limbajul de programare simplu LPS, pe baza căruia vom explica concepte și noțiuni importante în programarea calculatoarelor.

Proiectarea LPS a avut în vedere o familie de limbaje folosite în prezent de majoritatea programatorilor : limbajele de tip *pseudocod*. Ele nu sînt în realitate limbaje de programare, ci folosesc la descrierea algoritmilor într-o formă asemănătoare limbajului natural. Algoritmii descriși în pseudocod pot fi apoi cu ușurință transcriși într-un limbaj de programare (ALGOL, PASCAL, FORTRAN), transcrierea fiind aproape mecanică dacă se cunosc cele câteva reguli necesare.

Prezentarea limbajului LPS se va face prin intermediul sistemului de prelucrare a datelor simplu.

1.1. SISTEMUL DE PRELUCRARE A DATELOR SIMPLU (SPDS)

Vom prezenta în cele ce urmează un sistem de calcul cu o funcționare foarte simplă, sistem pe care îl vom numi pe scurt SPDS. *El nu există în realitate*, scopul prezentării fiind în primul rând didactic : plecând de la funcționarea lui vom introduce notiuni și concepte de bază în programarea sistemelor de calcul reale.

Structura și funcționarea SPDS au rezultat prin simplificări repetate ale structurii și funcționării sistemelor reale de prelucrare a datelor, reținând de la acestea numai aspectele importante pentru utilizator. Deci putem afirma :

Sistemul de prelucrare a datelor simplu (SPDS) este un *model* (o imagine simplificată) al sistemelor de calcul reale.

1.1.1. Structura și funcționarea SPDS

1. Structura SPDS este cea din figura 1.1.

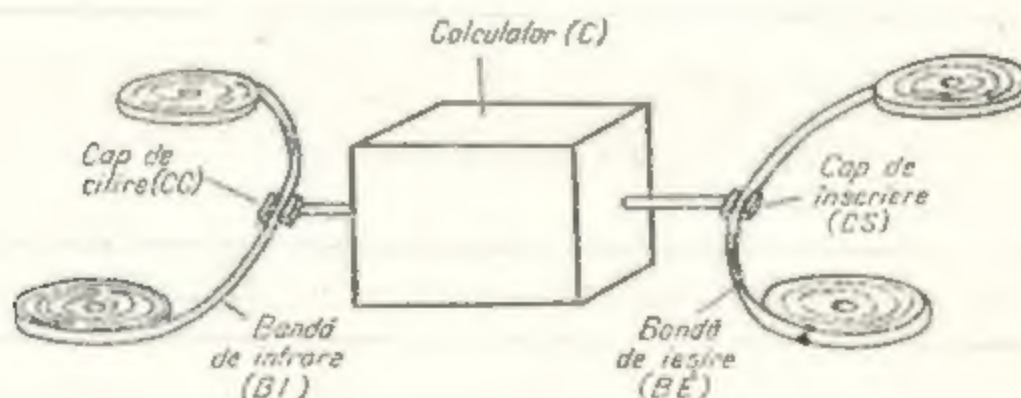


Fig. 1.1

SPDS este alcătuit din :

- banda de intrare BI,
- banda de ieșire BE și
- calculatorul C.

Vom descrie în continuare părțile componente ale SPDS și apoi funcționarea întregului ansamblu.

2. *Benzile de intrare și ieșire* (BI, respectiv BE) servesc la citirea datelor inițiale în vederea prelucrării lor de către calculatorul C și respectiv la înscriserea rezultatelor calculelor realizate de C.

BI și BE sînt modele ale fișierelor de intrare (pachetul de cartele perforate conținînd datele inițiale) respectiv de ieșire (succesiunea de linii scrise la imprimantă) ale sistemelor de prelucrare automată a datelor.

Succesiunea datelor conținute pe fiecare din cele două benzi va forma și în acest caz un fișier, *fișierul de intrare* și respectiv *fișierul de ieșire* ale SPDS.

Putem să ne facem o idee asupra acestor benzi gîndindu-ne la banda magnetofonului sau la banda de hîrtie a telegrafului. În plus, benzile BI și BE sînt divizate în celule, fiecare celulă conținînd o informație (dată) „depusă” în prealabil printr-un procedeu de înregistrare magnetică, tipărire etc. De altfel, trebuie spus că nu ne interesează în legătură cu aceste benzi natura suportului informației și nici modul de reprezentare a datelor, ci numai interpretarea lor. În acest sens :

Restrîngem interpretarea datelor înscrise pe benzi la următoarele tipuri :

- numere întregi,
- numere reale,
- valori logice,
- șiruri de caractere, numite și texte (le vom prezenta cuprinse între ghilimele sau apostrofuri).

În figura I.2 sînt prezentate cîteva exemple de tipuri de date.

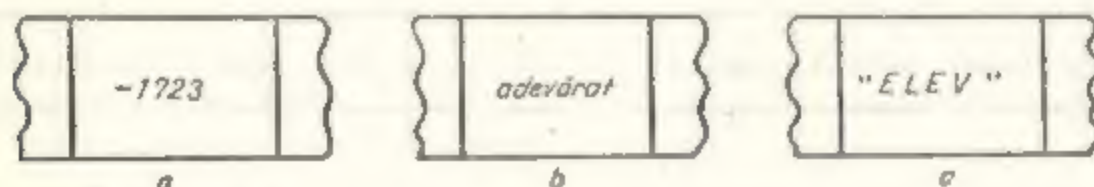


Fig. I.2

În cazul în care o celulă nu conține nici o dată din tipurile de date amintite, considerăm — prin convenție — că ea conține un caracter special numit „spațiu” sau „blanc” notat de obicei cu \square .

3.

Calculatorul C are următoarele părți componente :

- capul de citire CC,
- capul de înscrisere CS,
- memoria M,
- dispozitivul de prelucrare DP și
- dispozitivul de comandă DC.

Schema-bloc a calculatorului C este prezentată în figura I.3.

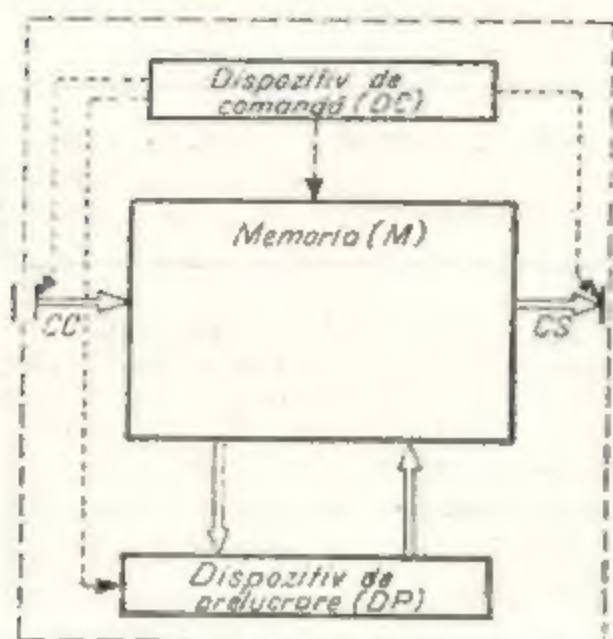


Fig. I.3

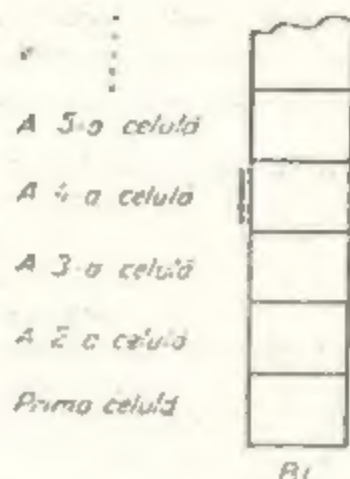


Fig. I.4

Schema arată, în afara componentelor amintite, transferul datelor între componente cu săgeată dublă și transferul comenzilor cu săgeată punctată. Vom detalia în continuare componentele lui C.

4. *Capul de citire CC* (asemănător celui de la magnetofon, de exemplu) folosește la citirea datelor aflate pe banda BI. Pentru aceasta CC este întotdeauna poziționat exact pe o celulă a benzii numită *celulă curentă*. După citirea datei aflată în celulă curentă, banda se deplasează într-un singur sens astfel încât sub capul de citire să ajungă celula următoare care devine noua celulă curentă.

Avansul benzii se face sacadat, celulă cu celulă, într-un singur sens.

Pentru a ne putea referi la celulele benzii BI le vom numerota. La fiecare avans numărul celulei curente crește cu 1. În figura I.4 se prezintă simplificat un caz în care celula curentă este a patra celulă a benzii BI.

5. *Capul de înregistrare CS* (asemănător capului de înregistrare a benzii magnetice la magnetofon sau dispozitivului ce imprimă sumele de plată pe banda de hârtie a mașinii electromecanice de casă aflată în multe din magazine) folosește la depunerea datelor pe banda BE. Considerentele prezentate privind celula curentă și deplasarea celulei cu celulă ale benzii BI sînt valabile și pentru capul de înregistrare și banda BE. Convențional, vom considera că celulele benzii BE care nu au trecut încă de capul de înregistrare conțin caracterul blank.

Starea capetelor este dată de numărul ce reprezintă poziția lor pe benzile respective.

6. *Funcționarea SPDS* va începe întotdeauna avînd capetele CC și CS poziționate pe primele celule ale benzilor respective.

În continuare, calculatorul C va citi date de pe BI, le va prelucra (după cum vom vedea mai departe) și, din cînd în cînd, va scrie valori (date) pe BE. În timpul funcționării SPDS, cele două benzi se vor deplasa în sens unic, în mod independent una de alta.

La un moment dat funcționarea SPDS va înceta. Șirul de date citite de pe BI pînă la acel moment formează *datele de intrare* ale prelucrării realizate de SPDS, iar șirul de date înscrise pe BE pînă la acel moment formează *rezultatele prelucrării*.

Funcționarea SPDS într-o prelucrare are ca rol transformarea datelor de intrare în rezultate.

Funcționarea poate fi văzută și ca o transformare a *fișierului de intrare* în *fișierul de ieșire*.

1.1.2. Funcționarea calculatorului C

Să vedem cum se realizează prelucrările în cadrul calculatorului C. Pentru aceasta vom analiza, mai întîi, structura și funcționarea celorlalte componente ale calculatorului: memoria, dispozitivul de prelucrare și dispozitivul de comandă.

1. Memoria M este alcătuită dintr-o mulțime de celule ce pot găzdui date din cele patru tipuri menționate. Putem să ne formăm imagini intuitive asupra memoriei gîndindu-ne la un fagure de albine, la un dulap cu multe sertare identice sau la casetele pentru bagaje din gări. Fiecare celulă a fagurelui sau fiecare sertar al dulapului ar putea fi privite ca celule ale memoriei M. În asemenea celule pot fi înscrise date sau din ele pot fi extrase date. Dacă vrem să înscriem o dată într-o celulă ce conține deja una, noua dată va lua locul celei vechi, cea veche dispărînd definitiv din celulă. În schimb, extragerea unei date dintr-o celulă a memoriei nu modifică conținutul celulei, operația executîndu-se prin copierea valorii.

Într-o celulă a memoriei nu poate exista decît cel mult o dată (de tip numeric, logic sau șir de caractere¹). În celulele memoriei pot fi „înscrise” date și din celule pot fi „citite” date.

O celulă în care nu s-a înscris nici o astfel de dată se numește neinițializată și orice încercare de a o citi este o eroare pe care calculatorul o semnalează.

¹ La calculatoarele reale capacitatea unei celule de memorie este limitată, ceea ce afectează uneori serios prelucrările. Calculatorul C fiind imaginar, ne putem permite să nu-i limităm capacitatea celulelor memoriei (de exemplu, să memorăm orice număr sau orice șir finit de caractere).

Pentru a putea citi și scrie din/în celulele memoriei este necesar să ne putem referi la ele. În limbajele de programare, deci și în LPS, referirea la celulele memoriei se face printr-un nume simbolic, numele celulei. În cadrul unei prelucrări necesitate de rezolvarea unei probleme folosind SPDS, un număr finit de celule din M capătă fiecare câte un nume pe care și-l păstrează în tot cursul prelucrării. Ansamblul acestor celule formează *zona datelor* prelucrării. În cele ce urmează vom scrie numele celulelor din zona datelor cu litere mici: x , α , b , max ș.a.m.d. În reprezentările grafice ale memoriei, cum este cea din figura I.5, celulele din zona datelor vor fi reprezentate compact, fiecărei celule asociindu-i-se conținutul, numit *valoarea* celulei, și numele ei, numit uneori și *adresa* sau *referința* celulei.

Starea memoriei în cursul unei prelucrări este dată de mulțimea perechilor (adresă ; valoare) ale tuturor celulelor zonei de date.

Pentru zona datelor din figura I.5 starea memoriei este dată de mulțimea

$\{(x : -15), (\alpha : \text{fals}), (max : 153), (nume : 'ION'), (val : 54,7), (m : 10),$
 $(log : \text{adev\textbf{ă}rat}), (cod : 'F5'), (an : 1980)\},$

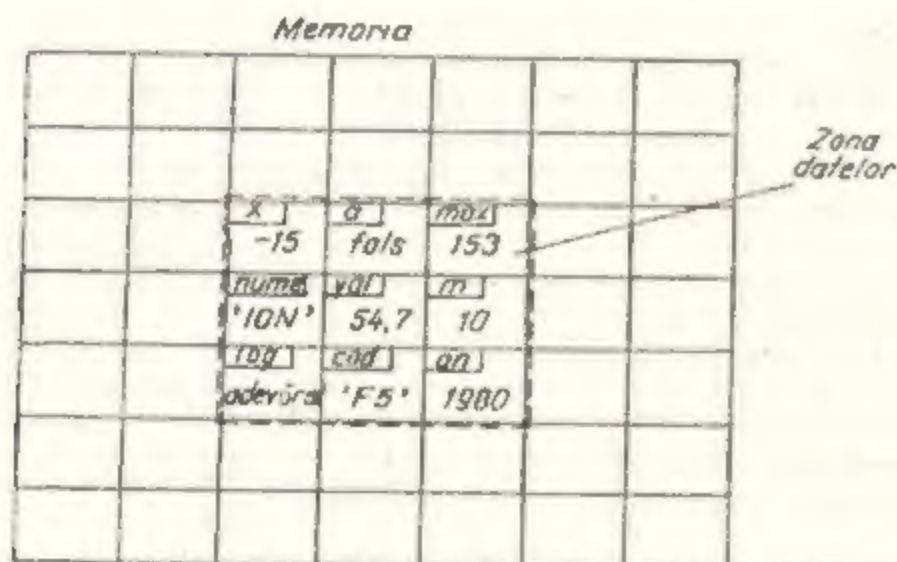


Fig. I.5

2. *Dispozitivul de prelucrare* DP realizează calculul valorii unei funcții pentru care se cunosc valorile argumentelor. DP este reprezentat schematic în figura I.6. Valorile argumentelor reprezintă „intrarea” în DP, iar valoarea funcției „ieșirea” din DP.

În cele ce urmează vom considera că funcțiile realizate de DP pot fi descrise prin expresii ce conțin operații cunoscute, cum ar fi :

a) operații algebrice : adunare, scădere, înmulțire, împărțire și ridicare la putere,

b) operații logice : negație, conjuncție și disjuncție,

c) compararea a două valori numerice în scopul stabilirii unei valori logice (adev\textbf{ă}rat sau fals) după cum cele două valori sînt sau nu într-una din relațiile : $<$, \leq , $=$, \neq , \geq , $>$.

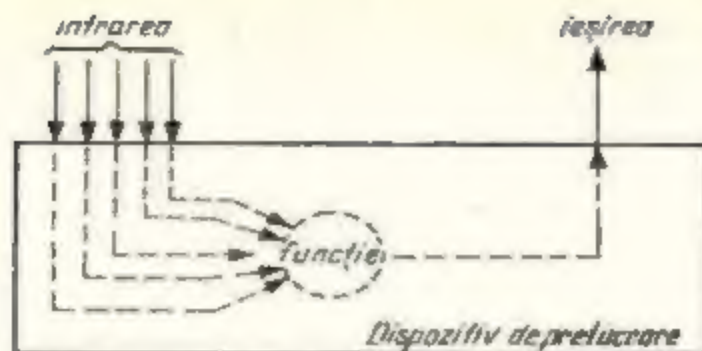


Fig. 1.6

În fine, expresiile ce descriu funcțiile realizate de DP pot conține și funcții trigonometrice, exponențiale, logaritmice etc. (în general, funcții elementare).

Realizarea calculului valorii unei funcții de către DP este bine să fie privită ca o secvență de operații de tipurile a), b), c), precum și evaluări de funcții cunoscute deja amintite. Vom exemplifica acest mod de a privi activitatea dispozitivului de prelucrare pe două cazuri prezentate schematic în figurile 1.7, a și 1.7, b.

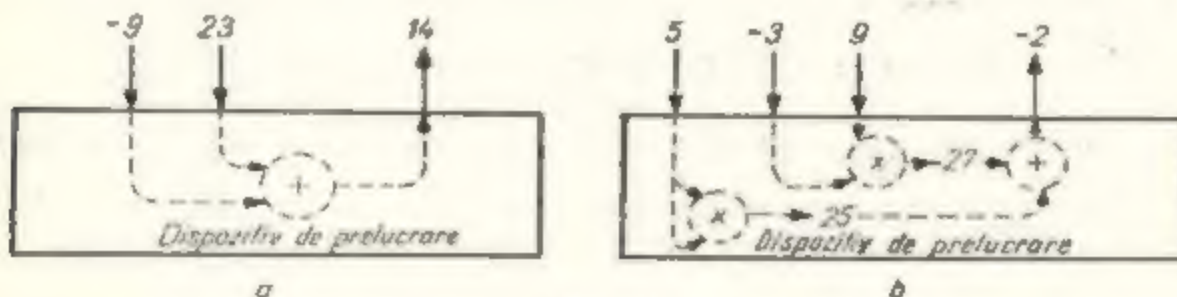


Fig. 1.7

În figura 1.7, a se prezintă evaluarea unei funcții $f(x, y) = x + y$ pentru intrările $x = -9$ și $y = 23$. Valoarea funcției, 14, rezultă din operația de adunare a celor două valori: -9 și 23.

În figura 1.7, b se prezintă evaluarea unei funcții $g(x, y, z) = x \cdot x + y \cdot z$ pentru intrările $x = 5$, $y = -3$, $z = 9$. Se observă că DP realizează o secvență de trei operații. Ordinea operațiilor este impusă. Ea rezultă din regulile binecunoscute de evaluare a expresiilor algebrice: înmulțire, din nou înmulțire și la urmă adunare. Rezultatele intermediare ale unor operații intervin, după cum se vede, în operații ulterioare.

În figura 1.8 se prezintă evaluarea unei funcții $h(x, y) = (x + y > 0) \wedge (y \neq 5)$ a cărei valoare este fals pentru cazul $x = -7$, $y = 3$. Din nou

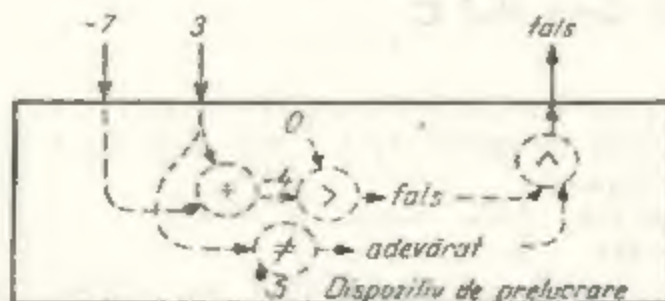


Fig. 1.8

regula de evaluare a expresiilor algebrice pe baza priorităților operațiilor asigură o secvență unică de operații: adunare, comparare la $>$, comparare la \neq , conjuncți. În fine, expresia conține două constante: 6 și 5 ale căror valori considerăm ca sînt asigurate de către DP, astfel încît doar valorile argumentelor să constituie „intrarea” lui DP.

3. *Deputatul de comandă DC* este locul în care se ia n decizie asupra modului de funcționare a calculatorului C. Pe baza acizilor lui C, DC comandă executarea unei operații din setul celor patru operații de baza ale calculatorului.

Operațiile de baza ale calculatorului C sînt: citirea, atribuirea, scrierea și oprirea.

a) Efectuarea unei operații de *citire* a valorii celulei curente de pe BI și depunerea ei într-o celulă dată din M necesită din partea DC comenzi pentru executarea următoarelor acțiuni:

- extragerea de către CC din celula curentă a lui BI a informației conținute de aceasta,
- plasarea valorii în celula din M precizată în operație (deci: *modificarea valorii celulei respective*),
- avansul cu o celulă a lui BI (deci: *modificarea poziției CC*).

b) Efectuarea unei operații de *atribuire* a valorii unei funcții date unei celule din M, a cărei semnificație, necesită următoarele acțiuni comandate de DC:

- copierea valorilor din celulele lui M care reprezintă argumentele funcției,
- evaluarea funcției de către DP folosind valorile argumentelor extrase din M,
- înscirerea valorii rezultatului în celula din M precizată în operație (deci: *modificarea valorii celulei din M*).

c) Efectuarea unei operații de *scriere* a unei valori dintr-o celulă dată din M în celula curentă de pe banda BE necesită următoarele acțiuni comandate de DC:

- copierea din celula dată a lui M a valorii conținute
- înscirerea valorii în celula curentă a benzii BE (deci: *modificarea conținutului celulei curente a lui BE*),
- avansul cu o celulă a lui BE (deci: *modificarea poziției CS*).

d) Operația de *oprire* are ca efect blocarea funcționării calculatorului C de către DC.

1.1.3. Starea calculatorului C

Din cele arătate pîna aici rezultă că dacă dorim să cunoaștem *starea calculatorului* la un anumit moment al funcționării lui în cadrul unei aplicații, este suficient să cunoaștem:

- a) poziția CC pe BI (deci *starea CC*),
- b) poziția CS pe BE (deci *starea CS*),
- c) zona datelor aplicației ca mulțime a perechilor (adresă, valoare), deci *starea memoriei*.

În figura I.9 se prezintă două stări distincte ale SPDS despărțite de execuția unei operațiuni de citire în celula 4 a memoriei. Se observă de la figura I.9, a la figura I.9, b creșterea cu 1 a poziției CC (de la 4 la 5) și modificarea conținutului celulei x cu valoarea existentă în celula curentă (a 4 a) citită. În figura I.9, a starea calculatorului C este data de tripletul

$$(4, 1, \{(i; 4), (x; 2), (m; 13)\})$$

iar în figura I.9, b de tripletul:

$$(5, 1, \{(i; 4), (x; 24), (m; 13)\}).$$

Operațiunile de bază citire, atribuire și scriere modifică starea calculatorului astfel:

- a) citirea modifică poziția capului de citire și valoarea celulei din M afectată de citire;
- b) atribuirea modifică valoarea celulei din M unde se dă pune rezultatul;
- c) scrierea modifică poziția capului de înscriere.

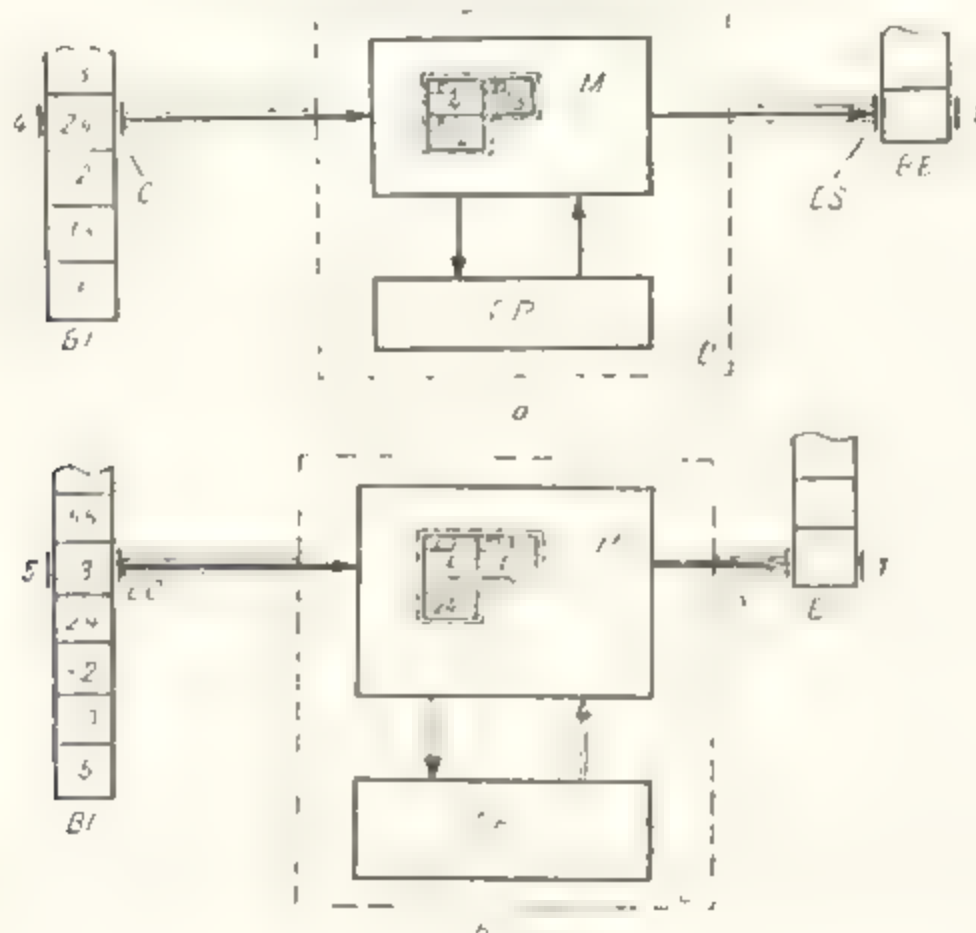


Fig. I.9

Se observă că adăugând în starea lui C conținutul bezzilor BI și BE obținem un sistem închis a cărui succesiune a stărilor, numită și evoluție, depinde doar de șirul operațiilor prin care obligăm sistemul să treacă. Păstrând mereu un șir de stări și operațiilor și modificând pentru fiecare nouă evoluție conținutul bezzilor de intrare BI obținem la sfârșitul evoluțiilor (atunci când se execută operații de oprire) bezzii de ieșire cu conținuturi diferite (rezultatele). Totalitatea perechilor (date de intrare, rezultate) caracterizează, după cum vom vedea în paragraful următor, șirul operațiilor de executat.

1.1.4. Programarea calculatorului C

1. Am denumit *date de intrare* șirul datelor aflate în celulele citite ale BI și *rezultate* șirul datelor aflate în celulele scrise ale BE la sfârșitul prelucrării (după execuția operației de oprire). Transformarea datelor de intrare în rezultate este consecința execuției unei secvențe de operații de bază.

De obicei, mulțimea datelor de intrare pentru care putem executa o anumită secvență de operații este dată prin specificarea problemei. Să notăm cu D această mulțime numită și *domeniul* secvenței de operații.

Să considerăm și mulțimea tuturor perechilor (date de intrare, rezultate) pentru datele de intrare din D . Să notăm cu K această mulțime. Ea se numește *comportare externă* a secvenței de operații pentru domeniul D . Comportarea externă este a doua componentă a specificării oricărei probleme.

Dându-se o pereche (D, K) , a *programa calculatorul C* înseamnă a determina o secvență de operații de bază care, executate în ordinea secvenței pentru toate datele din D , să aibă mulțimea K drept comportare externă.

2. Să considerăm mulțimile :

$$D = \{(x, y) \mid x, y \in \mathbf{R}\}$$

$$K = \{((x, y), z) \mid (x, y) \in D, z = \max(x, y)\}.$$

Se poate observa că nu există nici o secvență de operații de bază care să scrie pe BE, pentru orice pereche de numere reale, pe cel mai mare dintre ele. Ar trebui doua secvențe : una care să scrie primul număr citit, cealaltă care să scrie al doilea număr citit. Calculatorul va executa una sau alta din cele două secvențe în funcție de mărimile relative ale celor două numere. Asemenea operații, care să aleagă între mai multe secvențe de operații pe una singură în funcție de îndeplinirea anumitor condiții, nu au fost încă definite pentru calculatorul C. Ele există însă în cazul calculatoarelor reale și sînt strîns legate de conceptul de *program memorat*.

În cele ce urmează vom introduce și pentru calculatorul C, posibilitatea de memorare a programului, mărind-i astfel autonomia de funcționare.

Din prezentarea calculatorului C nu a rezultat de unde primește comenzile pe care le execută. Nu ne-a interesat acest aspect, deoarece ne-am îndreptat atenția spre ceea ce poate face calculatorul din punct de vedere al prelucrării, acest lucru fiind descris prin secvențe de operații de bază.

Pentru a comanda calculatorul C, cel mai simplu mod ar fi să-i dăm comandă după comenzi prin intermediul unor butoane, comutatoare etc. În felul acesta au fost comutate primele calculatoare, iar astăzi sînt comutate unele calculatoare de birou și unele mașini de calculat de birou. Într-o astfel de situație omul urmărește desfășurarea calculului pas cu pas și, în funcție de rezultatele intermediare, ia decizii asupra modului în care continuă prelucrarea, deci asupra comenzilor următoare. Se știe că această modalitate este dezavantajoasă din punct de vedere al vitezei de calcul deoarece nu se exploatează viteza cu care calculează calculatorul, mult mai mare în raport cu viteza de reacție a omului.

Înzestrarea calculatorului cu posibilitatea de a decide asupra următoarei operații de executat necesită „cunoașterea” de către calculator a tuturor posibilităților de continuare a calculului, deci cunoașterea de la început a tu-

turor calculelor ce trebuie să le execute calculatorul în diversele situații posibile, cu alte cuvinte a întregului program. Ajungem astfel la un concept extrem de important: *programul memorat*. Calculatoarele moderne funcționează aproape în exclusivitate „comandate” de un program memorat.

Programul este memorat într-o „zonă a programului” aflată în memoria calculatorului, zonă distinctă în cazurile reale doar din punct de vedere al modului de interpretare a informațiilor din ea. Dispozitivul de comandă își extrage din zona respectivă instrucțiunile una după cealaltă, conform unor reguli bine stabilite.

În cazul calculatorului C putem completa schema din figura 1.3 cu o *memorie a programului MP* în care utilizatorul depune, printr-un procedeu oarecare de încărcare, instrucțiunile programului (fig. 1.10). Nu ne interesează cum este organizată memoria MP, cum sunt reprezentate instrucțiunile și nici cum dispozitivul DC își extrage de aici următoarea instrucțiune de executat.

Ceea ce este important pentru noi este cunoașterea acelor operații ale calculatorului C care permit schimbarea locului din MP de unde DC își extrage următoarea instrucțiune de executat. Aceste operații sunt altele decât cele de bază. Ele se referă la dirijarea în memoria MP a indicatorului următoarei instrucțiuni de executat. Ele se numesc *operații de control*. Rolul operațiilor de control va fi prezentat în paragraful 1.3 cînd se vor descrie instrucțiunile condiționate și de ciclare.

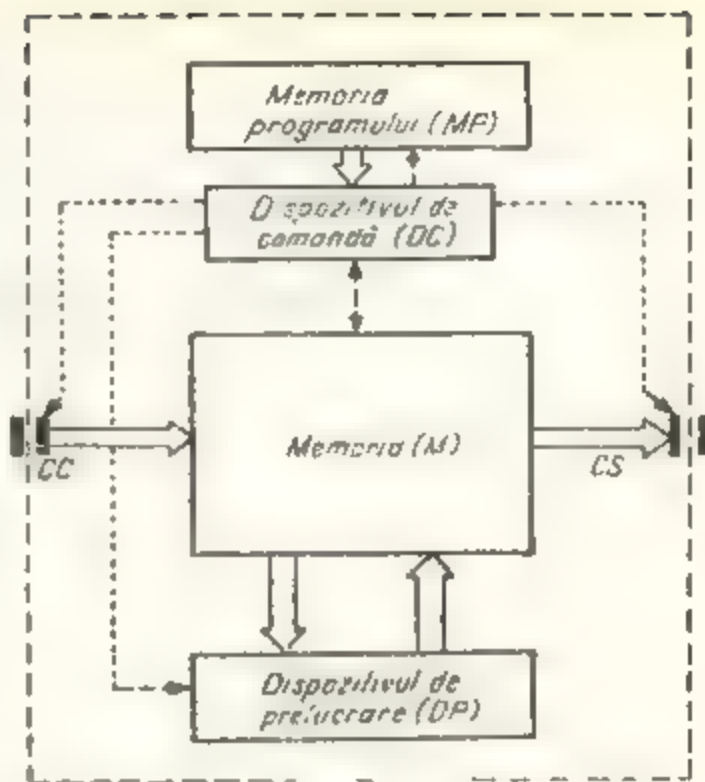


Fig. 1.10

INTREBĂRI

1. Care sînt componentele SPDS? Cum sînt alcătuite benzile de intrare și ieșire?
2. Cum funcționează SPDS?
3. Care sînt componentele calculatorului C și care sînt relațiile între ele?
4. Cum funcționează calculatorul C?
5. Care este structura memoriei M? Ce conțin celulele memoriei și cum are loc accesul la ele?
6. Cum funcționează dispozitivul de prelucrare DP? Care sînt operațiile realizate de DP?
7. În relația memorie-dispozitiv de prelucrare, memoria este elementul pasiv, cel care suportă operațiile, iar dispozitivul de prelucrare este elementul activ, cel care realizează prelucrările aritmetico-logice. Cum se realizează această relație?
8. Cum își manifestă dispozitivul de comandă DC funcția de conducere a activității calculatorului C?

9. Care sînt operațiile de bază ale calculatorului C?
10. Cum poate fi comandată din exterior o mașină de calcul?
- 11*. Care sînt avantajele programului memorat, ce implicații are acest concept asupra autonomiei de lucru a calculatoarelor?
- 12*. Care sînt informațiile de stare ale SPDS și ale calculatorului C? Descrieți operațiile de bază în termenii modificării informațiilor de stare.

1.2 REPREZENTAREA ALGORITMILOR ÎN LIMBAJUL LPS

1.2.1. Definirea limbajelor de programare

Limbajele de programare se pot învăța cum se învăță și alte limbi artificiale¹ limbajul formulelor și expreselor matematice, limbajul logicii, limbajul formulelor și ecuațiilor chimice, limbajul schemelor circuitelor electrice etc.

În mod special, în cazul limbajelor de programare, definirea lor trebuie să fie precisă. Prin analogie cu limbajul natural² un program este o „propoziție” într-un limbaj de programare.

El descrie o anumită funcționare a calculatorului. Care este această funcționare rezultă din interpretarea programului respectiv conform definiției limbajului. Definiția trebuie deci să fie clară pentru ca *orice* program să fie înțeles la fel de *orice* utilizator și anume exact în sensul pe care calculatorul îl atribuie programului respectiv.

Și în cazul limbajelor de programare, ca și în cazul limbajelor naturale, definirea se face pe părți componente. Se știe că în gramatica unui limbaj natural găsim reguli de forma: o frază este alcătuită din una sau mai multe propoziții, o propoziție este formată din subiect, predicat și eventual alte părți de propoziție etc.

Vom regăsi deci și la limbajele de programare asemenea „părți de propoziție”. Despre una din ele deja am amintit: instrucțiunea. Se știe că un program este format din instrucțiuni.

Vom numi *elemente componente ale limbajului* acele construcții din limbaj caracterizate de anumite particularități comune de scriere și semnificație.

De exemplu, în limba română noțiunea de predicat se caracterizează atât la nivelul structurii (cum este alcătuit) — un verb sau un verb cu un nume predicativ, cât și la nivelul semnificației — exprima o acțiune etc. Într-un lim

Exercițiile cu un grad sporit de dificultate au fost notate cu *. Dificultatea poate proveni din mai multe cauze:

- răspunsul necesită cunoștințe din clasele precedente,
- răspunsul presupune o discuție asupra soluțiilor posibile
- răspunsul necesită aprofundarea unor cunoștințe de bază

¹ Limbaj artificial — Limbaj creat deliberat de un grup restrâns de oameni pentru comunicarea într-un anumit domeniu al științei sau tehnicii.

² Limbaj natural — Limbaj apărut spontan, de-a lungul istoriei, în mari colectivități umane în cadrul procesului de muncă în vederea comunicării între membrii colectivităților respective.

ba) de programare, o instrucțiune are reguli precise de scriere, iar semnificația ei constă dintr-un anumit efect pe care îl are execuția instrucțiunii de către un calculator asupra stării acestuia.

Cele două niveluri — cel al scrierii și cel al semnificației, apar foarte bine conturate în cazul limbajelor de programare.

Orice element component al unui limbaj de programare trebuie definit la două niveluri :

- 1° cum se formează (serie) corect (*nivelul sintactic*),
- 2° ce semnificație are pentru utilizator, semnificație înțeleasă ca efect produs asupra funcționării unui calculator (*nivelul semantic*).

Regulile de formare a programelor corecte alcătuiesc *sintaxa* limbajului de programare.

Regulile ce determină semnificația programelor corecte alcătuiesc *semantica* limbajului de programare.

În cele ce urmează vom prezenta un limbaj folosit pentru descrierea algoritmilor : limbajul LPS.

Limbajul LPS va fi folosit ca o fază intermediară în scrierea programelor, alternativă posibilă, de cele mai multe ori pe fațeta, schemelor logice.

1.2.2. Instrucțiunile de bază ale limbajului LPS

1. Vom introduce întâi cele mai simple instrucțiuni ale LPS. Ele corespund ca semnificație operațiilor de bază ale calculatorului C : citirea, scrierea, atribuirea și oprirea. Pentru a da o reprezentare, o descriere simbolică acestor operații, să ne imaginăm că putem comanda calculatorul C prin comenzi scrise : citește !, scrie ! etc. Sub această formă comenzile apar însă incomplete. În cazul comenzii „citește !”, de exemplu, calculatorul C știe de unde să citească (de pe banda BI) și ce să citească (valoarea din celula curentă), dar nu va ști ce să facă cu valoarea citită, nu va ști în care celulă din M să o plaseze. De aceea ord. nul trebuie completat cu numele celulei în care se depune valoarea citită

citește x

Pentru a marca mai bine partea fixă a comenzii, cuvântul „citește”, o vom scrie îngroșat (cu litere aldine) astfel :

citește x

În fine, deoarece în locul lui x poate fi y sau *alfa* sau *max* sau orice alt nume de celulă din memorie, vom defini scrierea corectă a comenzii sub forma :

citește < variabilă >

De reținut următoarele convenții folosite în această definiție :

Cuvîntul *<variabilă>*¹ scris cu litere cursive între parantezele unghiulare *< și >* desemnează elemente ale unei mulțimi (mulțimea numelor celulelor memoriei M). În acest fel, *<variabilă>* din punct de vedere sintactic este „un cuvînt cu litere mici”, iar semantic „o celulă din memoria calculatorului”.

Vom numi *instrucțiune* o reprezentare simbolică a unei comenzi date calculatorului. În cazul de față instrucțiunea descrisă se numește *instrucțiune de citire*. Dacă *a*, *beta*, *alfa* sînt variabile, atunci :

citește *a*

citește *beta*

citește *alfa*

sînt instrucțiuni de citire.

În figura 1.11 se prezintă efectul instrucțiunii **citește *a*** asupra calculatorului C în cazul în care celula curentă a benzii de intrare conține numărul 15. Se observă că efectul este cel al operației de citire a calculatorului C.

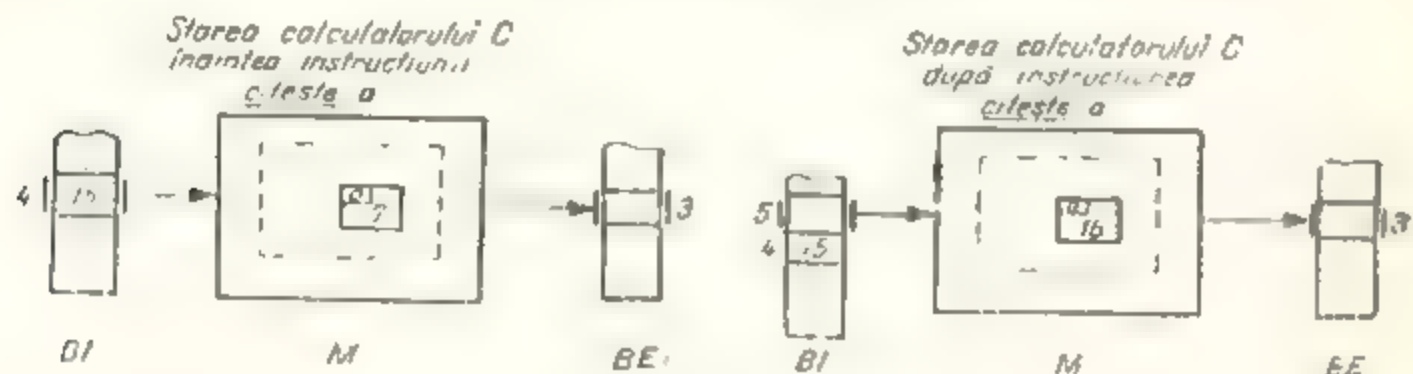


Fig. 1.11

2. Asemănător, instrucțiunea de scriere din LPS, ca „ordin scris” adresat calculatorului C, o vom nota astfel :

scrie *<variabilă>*

Deci **scrie *p*** sau **scrie *limită*** vor fi instrucțiuni de scriere corecte în LPS. Efectul unei instrucțiuni de scriere asupra calculatorului C este cel al operației de scriere. De exemplu, în figura 1.12 se prezintă efectul instrucțiunii **scrie *p*** cînd în celula *p* din memoria calculatorului se află valoarea 13.

¹ Denumirea de variabilă provine din limbajul expresiilor algebrice. O variabilă este un nume (literă sau cuvînt, cu indici sau fără) ce desemnează orice valoare dintr-o mulțime. Mulțimea formență domeniul variabilei respective. De exemplu, în expresia $x^b + 1$, $x \in \mathbb{R}$, x este o variabilă al cărei domeniu este mulțimea numerelor reale. În cazul limbajelor de programare, variabilele desemnează întotdeauna o pereche celulă-valoare memorată, celula rămînînd mereu aceeași în cursul execuției unui program, în timp ce valoarea memorată se poate schimba, dar rămînînd într-o aceeași mulțime dată de tipul variabilei.

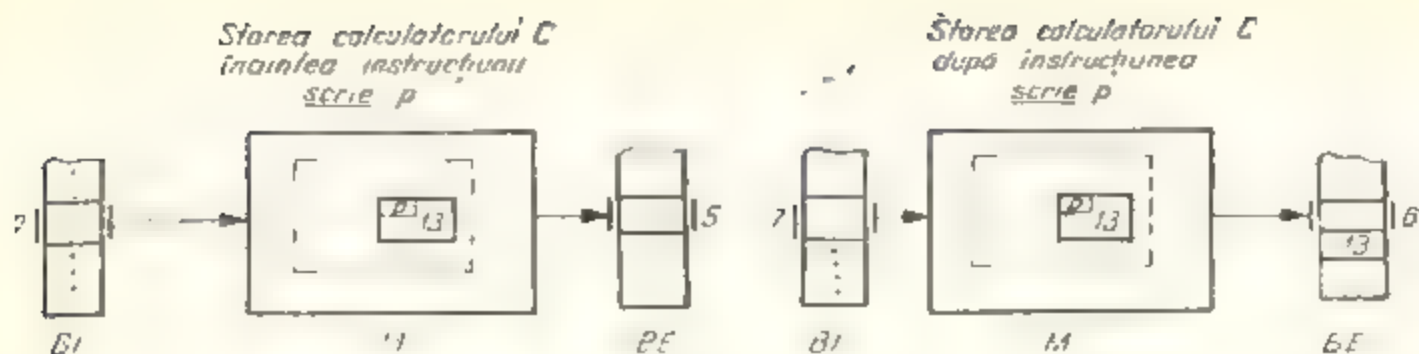


Fig. I.12

3 Pentru a efectua o operație de atribuire vom comanda calculatorul C cu :

atribuie *<variabilă>* \leftarrow *<expresie>*

unde *<variabilă>* reprezintă numele celulei din memorie în care se depune rezultatul calculului descris prin *<expresie>*.

De exemplu

atribuie *a* \leftarrow 5

atribuie *vit* \leftarrow *sp/limp*

atribuie *x* \leftarrow $(-b + \sqrt{\text{delta}})/(2 \cdot a)$

atribuie *y* \leftarrow $(\exp(x) - \exp(-x))/2$

atribuie *alta* \leftarrow $\arcsin(\sin^2(x) - \cos^2(x)) - 1$

sînt instrucțiuni de atribuire. Să observăm :

a) **atribuie** și \leftarrow sînt părțile fixe ale instrucțiunii, *<variabilă>* și *<expresie>* au reprezentări diverse, în funcție de necesități.

b) sintaxa pentru *<expresie>* este cea a expreselor algebrice în care apar variabile și constante, cu operații de adunare, scădere, înmulțire și împărțire, cu evaluări de funcții cunoscute etc.

Să remarcăm în plus semnul \cdot pentru înmulțire, / pentru împărțire, notația *exp* pentru funcția exponențială și utilizarea exclusivă a parantezelor (.) în expresii.

În figura I.13 se prezintă modificările din memoria calculatorului C la primirea „ordinului” : **atribuie** *delta* \leftarrow $b^2 - 4 \cdot a \cdot c$

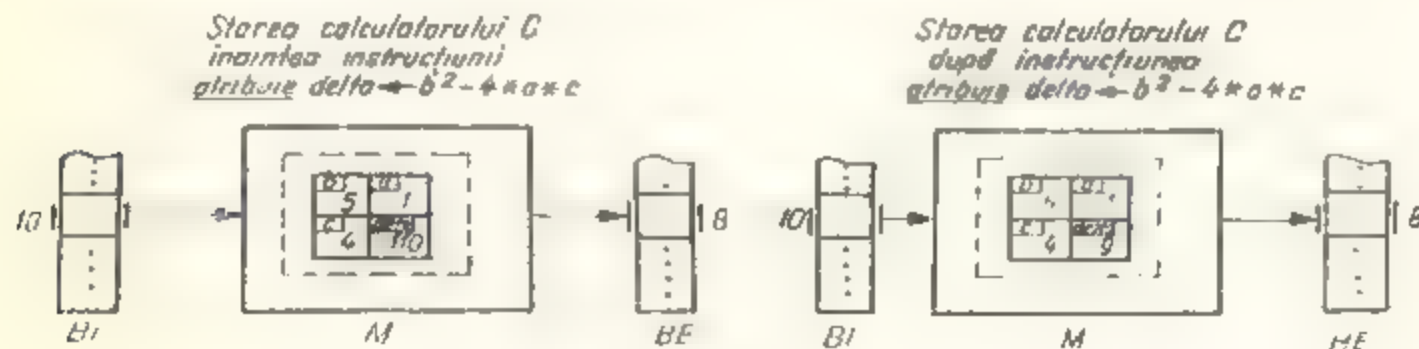


Fig. I.13

Putem să folosim instrucțiunea de atribuire și pentru valori logice. De exemplu: **atribuie $l \leftarrow$ adevărat**

atribuie $x \leftarrow w$ (dacă x și w conțin valori logice)

În general, expresiile folosite pot descrie calcule atât cu valori numerice, cât și logice. De exemplu, data celulele a și b conțin valorile 5 și respectiv 10, evaluarea expresiei $a^2 \leq b$: $a > b^2$ va produce ca rezultat **fals**.

4. Operația de oprire a calculatoarelui C o vom scrie în limbajul LPS :
stop

1.2.3. Secvența de instrucțiuni

1. După ce am văzut cum se scriu instrucțiunile de LPS în limbajul LPS, să vedem cum descriem în acest limbaj înfăptuirea operațiilor de bază ale calculatoarelor C, adică executarea succesivă. În mod obișnuit, în limbajul de programare ordinea normală de execuție a operațiilor este dată chiar de ordinea în care se scriu instrucțiunile.

Să luăm ca exemplu citirea a două valori succesive din BI și plasarea lor în memoria M în celulele a , respectiv b . În figura 1.14 sunt prezentate trei stări ale calculatoarelui C corespunzătoare celor 3 momente importante în desfășurarea execuției celor două citiri, între citiri și după executarea citirilor. În această figură folosim și o nouă reprezentare a stării sistemului SPDS. Ea

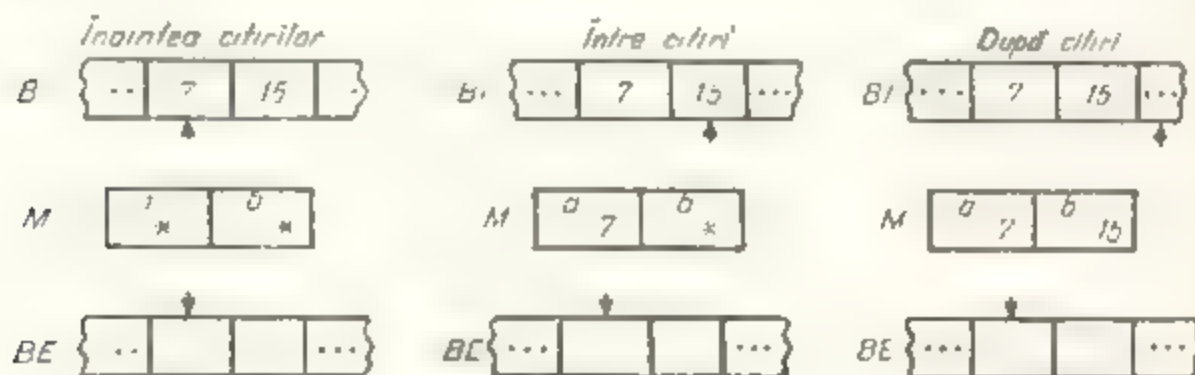


Fig. 1.14

este mai simplă decât cea folosită în figurile 1.11 - 1.13. Noua reprezentare cuprinde toate informațiile necesare :

- conținutul benzii BI (în partea superioară a figurii),
- conținutul benzii BE (în partea inferioară a figurii)
- conținutul memoriei M (la mijlocul figurii). Convenim ca în celulele neinițializate sau care conțin valori neinteresante din punct de vedere al prelucrării să plasăm semnul *.

pozițiile capetelor CC și CS (sigelele ce indică celule ale Lenzilor). Vom folosi în continuare această reprezentare. În cazul particular studiat, pe banda de intrare se afla două numere : 7 și 15. Cele două comenzi de citire se vor da evident în ordine : întâi citirea lui 7 în a , apoi citirea lui 15 în b . După cum se vede din figura, operația de citire în b pornește de la starea calculatoareului determinată de citirea în a .

Este natural ca ordinea executării operațiilor să o respectăm și în scrierea instrucțiunilor în LPS : întâi **citește a** , apoi **citește b**

Pentru a păstra claritatea programului LPS vom scrie cele două instrucțiuni una sub alta :

citește a

citește b

În asemenea grup de instrucțiuni scrise una după cealaltă îl numim *secvență de instrucțiuni*. Într-o secvență de instrucțiuni pot fi oricâte instrucțiuni (în particular chiar și una singură). De exemplu :

citește a

scrie a

este o secvență de instrucțiuni. Dacă dorim să știm ce producări determină această secvență va trebui să „înălțăm” efectele celor două instrucțiuni în ordinea scrierii lor, după ce s-a citit în celula *a* valoarea curentă de pe banda de intrare, se scrie pe banda de ieșire valoarea din celula *a* care este *citită* cea citită puțin mai înainte.

2. În cazul în care într-o secvență apar instrucțiuni de baza de același tip, consecutive vom introduce o notație simplificată scriind comanda **citește**, **scrie** sau **atribuie** o singură dată după cum vom înălța părțile lor variabile exact în ordinea în care au apărut în secvență. Deci :

citește a

citește b

le putem scrie :

citește a, b

(nu **citește b a** care este echivalent cu : **citește b**
citește a)

De asemenea :

scrie alfa

scrie beta

scrie omega

le putem scrie :

scrie alfa, beta, om. ca.

În fine, în cazul atribuirilor vom folosi semnul & („și” comercial) pentru a lega atribuiri succesive. De exemplu, secvența :

atribuie $b \leftarrow a + c$

atribuie $d \leftarrow a - c$

atribuie $e \leftarrow b^2 - d^2$

o putem rescrie compact astfel :

atribuie $b \leftarrow a + c$ & $d \leftarrow a - c$ & $e \leftarrow b^2 - d^2$

sau dacă expresiile sînt lungi,

{	atribuie	$b \leftarrow a + c$
		$\& d \leftarrow a - c$
	■	$\& e \leftarrow b^2 - d^2$

S-a marcat cu ■ plasat sub **atribuie** sfîșitul secvenței celor trei atribuiri. Arcul folosit între **atribuie** și ■ evidențiază o dată în plus unitatea celor trei instrucțiuni, faptul că ele formează o secvență de instrucțiuni de același tip pe care le-am scris pe mai multe rînduri.

Limbajul LPS nefiind un limbaj de programare propriu-zis, ci un limbaj de comunicare a algoritmilor, ne putem permite abateri în anumite limite de la regulile de scriere prezentate. Abaterile nu trebuie însă să afecteze semantica instrucțiunilor și nici să micșoreze claritatea descrierii algoritmului. După cum se va constata în exemplele prezentate în continuare, ne vom permite libertăți în scrierea unor elemente ale reprezentării algoritmilor în LPS cum ar fi expresiile, sau în compactarea secvențelor de instrucțiuni; nicideată însă nu vom afecta cu aceste abateri structura programului rezultată din incluziunea instrucțiunilor unele în altele, incluziune ce va apărea evidentă în cazul instrucțiunilor de control.

Algoritmul I

Ne propunem să descriem în limbajul LPS o prelucrare de date simplă realizată de calculatorul C.

Fie o bandă de intrare pe care se află două numere întregi. Să se descrie prelucrarea realizată de calculatorul C pentru ca cele două numere întregi să fie scrise în aceeași ordine pe banda de ieșire.

Algoritmul dorit trebuie să descrie în termeni celor 4 instrucțiuni de bază înlănțuirea operațiilor de bază ale calculatorului C care asigură realizarea prelucrării cerute.

Pentru aceasta, calculatorul va citi mai întâi numerele, memorându-le, și apoi le va înscrie pe BE. În LPS acest lucru se scrie astfel:

citește a

citește b

scrie a

scrie b

Figura I.15 prezintă cele 5 stări ale calculatorului C în cazul în care BI conține numerele 5 și -17.

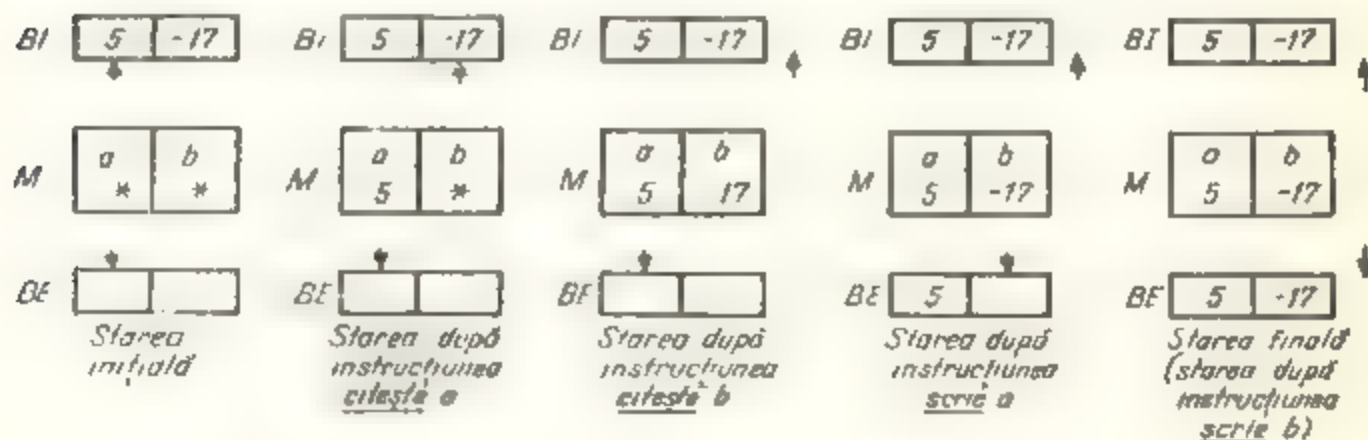


Fig. I.15

Observații Zona datelor pentru prelucrarea descrisă mai sus cuprinde două celule, a și b. Numere lor sînt tocmai variabilele folosite în secvență. Se observă că este neimportantă alegerea numerelor variabilelor. În general, este bine să le alegem cât mai sugestive pentru aplicația dată.

Să considerăm secvența :

citește a
scrie a
citește a
scrie a

zona datelor în prelucrarea descrisă de această secvență are o singură celulă, iar prelucrarea este aceeași cu cea descrisă de secvența de mai sus. Putem compara cele două prelucrări? Putem afirma despre una din ele că este mai bună decât cealaltă? Să observăm că a doua secvență descrie o prelucrare ce necesită mai puține celule de memorie decât prima. Deci, din punct de vedere al memoriei „folosite”, această prelucrare este mai bună decât prima. De multe ori însă, în căutarea unor prelucrări mai eficiente din punct de vedere al memoriei folosite, se poate pierde din claritatea descrierii.

1.2.4. Reprezentarea algoritmilor în LPS

1. În paragraful precedent am descris în LPS o prelucrare simplă folosind o secvență de patru instrucțiuni de bază. Pentru a fi o reprezentare completă a unui algoritm în LPS, cele patru instrucțiuni trebuie în primul rând completate cu o instrucțiune care să determine oprirea calculului de îndată ce rezultatele au fost obținute. Secvența, completată cu instrucțiunea de oprire, devine :

citește a
citește b
scrie a
scrie b
stop

În al doilea rând, reprezentarea completă necesită și o descriere a tuturor numerelor simbolice folosite în algoritm. Pentru a aprecia utilitatea unei asemenea descrieri să considerăm următoarea secvență :

atribuie $a \leftarrow 5$
atribuie $b \leftarrow 3,1$
atribuie $c \leftarrow a * b$
atribuie $d \leftarrow a \vee c$

Din primele două instrucțiuni rezultă că în celula a se memorează un număr întreg și în celula b un număr real. Trecând la a treia instrucțiune, să vedem ce valoare se calculează în vederea memorării ei în c . Este o valoare întreagă sau reală? Întrebarea nu este lipsită de sens deoarece valoarea rezultă din înmulțirea unui număr întreg cu unul real. Dacă în algebră este de la sine înțeles ce se întâmplă cu asemenea expresii mixte, în cazul calculatoarelor trebuie să fim atenți deoarece, după cum se știe, numerele reale și cele întregi au reprezentări interne diferite și, în consecință, operații de prelucrare diferite. O operație mixtă (cu operanzi de tipuri diferite) cum este cea de mai sus se va executa în două etape : întâi are loc o conversie de la reprezentarea valorii întregi a lui a la reprezentarea valorii reale echivalente, după care această valoare se înmulțește cu valoarea reală a lui b , rezultatul fiind o valoare reală.

În plus, într-un program acțiunea unui operator depinde și de felul operanzilor. Aceeași operație, să zicăm $+$ sau $/$, se realizează în \mathbb{Z} dacă operanzii sînt întregi și în \mathbb{R} dacă operanzii sînt reali, ceea ce conduce la rezultate diferite. Este deci important să cunoaștem ce fel de valori au operanzii fiecărei operații.

Cu a patra instrucțiune lucrurile se complică mai mult deoarece operația „sau logic” dintre a și c nu are nici un sens, valorile acestor variabile fiind numerice. În asemenea cazuri nu se poate pune problema conversiei implicite (ca mai sus, ci pur și simplu trebuie să o considerăm ca o eroare de programare.

„Detectarea” unor asemenea situații se poate face încă înainte de a executa programul dacă s-ar cunoaște pentru fiecare nume simbolic folosit mulțimea valorilor pe care acesta le poate reprezenta, deci domeniul lui de definiție. În cazul calculatorului C vom considera ca o variabilă se caracterizează, pe lângă nume și valoare conținută și prin domeniul de definiție, iar acest domeniu rămâne neschimbat pentru orice execuție a algoritmului. Pentru exemplul de mai sus, dacă a și c au valori numerice, doar dintr-o analiză a textului algoritmului, rezulta că expresia $a \vee c$ nu are sens fără a fi necesară execuția lui. În plus, specificarea domeniului de definiție sporește claritatea algoritmului, inteligibilitatea lui.

2. În toate limbajele de programare stabilirea domeniului de definiție al variabilelor se face prin intermediul *declarațiilor*. Conceptul de declarație este de maximă importanță pentru cei care învață să programeze calculatoare.

Declarațiile au ca efect asocierea la numele simbolice folosite în program a unor proprietăți numite *atribute*.

Numele simbolic folosit într-un program pentru a desemna o variabilă se numește *identificator*. În majoritatea limbajelor identificatorul este un șir de litere sau cifre în care primul caracter este literă. În cazul LPS am preferat excluderea cifrelor din scrierea identificărilor.

Domeniul de definiție a unei variabile desemnată printr-un identificator este astfel o mulțime de valori care rezultă din caracterizarea identificatorului prin atribute în cadrul declarațiilor. Sa prezentăm două atribute care sînt des folosite în limbajele de programare:

Tipul Majoritatea limbajelor au un număr de tipuri primitive, tipul întreg, tipul real, tipul logic, tipul caracter. Ele caracterizează mulțimi de valori cunoscute: \mathbf{Z} , \mathbf{R} , {adevărat, fals} și respectiv un alfabet de caractere. De exemplu:

real x

asociază identificatorului x tipul real, ceea ce înseamnă că x poate lua orice valoare din \mathbf{R} și numai din \mathbf{R} .

Asociat fiecărui tip există un set de operații de prelucrare a valorilor tipului respectiv. De exemplu, fiecărui tip numeric i se asociază cele patru operații: adunare, scădere, înmulțire și împărțire, rezultatul conservînd tipul operanzilor. Este evident că adunarea valorilor întregi va fi distinctă de adunarea valorilor reale datorită atât domeniilor de definiție distincte cît și reprezentărilor în calculator distincte. Tipului logic l se asociază de obicei cele trei operații logice etc.

În unele limbaje de programare apărute în ultimii ani sînt înglobate facilități ce permit programatorului să și creeze tipuri noi de date pe baza tipurilor primitive. Declarația unui nou tip de date într-un limbaj de programare necesită și definirea operațiilor tipului respectiv.

Structura Atributul de structură caracterizează numărul și organizarea elementelor componente ale unei variabile. Pînă acum toate variabilele întîlnite aveau un singur element. O variabilă cu o asemenea „structură” se mai numește *variabilă simplă*. În afara ei, cele mai cunoscute struc-

turi ale variabilelor sînt *tablourile* și *înregistrările*. În primul caz, o valoare a unei variabile-tablou este formată din valorile de același tip ale componentelor ordonate după unul sau mai multe criterii asemănător vectorilor sau matricelor. Se mai numesc structuri *omogene*. În al doilea caz, o valoare a unei variabile-înregistrare este formată din mai multe valori, de regulă diferite ca tip. Acestea sînt structuri *neomogene*.

Orice tip de structură trebuie să aibă asociat și un mecanism prin care să ajungă la componentele structurii. În cazul tablourilor, mecanismul este *indexarea*. Să presupunem o structură de tip tablou în care elementele componente (toate de același tip) sînt ordonate după un anumit criteriu (asemănător componentelor unui vector). Deci vom putea vorbi de primul element, de al doilea etc. Dacă v este numele variabilei atunci pentru a ne referi la al 5-lea element vom folosi expresia $v(5)$. Valoarea dintre paranteze se numește *indice* și servește deci la localizarea elementului căutat în cadrul structurii. Întocmai expresie desemnează o *variabilă indexată*.

Dacă ordonarea elementelor structurii se face după mai multe criterii vom avea atîția indici cîte criterii sînt. În cazul unei structurii de tip matrice vom avea doi indici: indicele de linie și indicele de coloană. Să presupunem o structură de acest fel numită *mat*. Expresia *mat* ($i + 1, j$), dacă valoarea lui i este 6, desemnează elementul aflat pe linia 7 și coloana 3. Din acest exemplu se observă că valorile în linii pot rezulta și din expresii mai complicate.

În general, un tablou trebuie caracterizat prin numărul de criterii de ordonare, numite *dimensiuni*, iar pentru fiecare dimensiune de valori pe care le pot lua în linie precum și prin tipul elementelor componente. De exemplu :

<i>mat</i>	tablou, 2 dim. (15, 20)	val. reale
------------	----------------------------	------------

este descrierea în LPS a unui tablou de valori reale cu numele *mat*, cu două dimensiuni pentru care indicii iau valori de la 1 la 15, și, respectiv, de la 1 la 20.

Vom relua și exemplifica cele de mai sus în I.3.3.

În cazul înregistrărilor, în mecanismul de acces se folosesc *selectori* care sînt nume simbolice asociate componentelor. Să presupunem o înregistrare declarată cu numele *data* avînd trei componente (cîmpuri) de tip întreg denumite *zi*, *lună* și *an* :

data = înregistrare (întreg *zi*, întreg *lună*, întreg *an*)

și o declarație care asociază numelui x structura neomogenă *data* :

data x

Dacă dorim să avem acces la o componentă vom folosi selectorii *zi*, *lună*, *an* astfel :

$x \cdot zi$, $x \cdot luna$, $x \cdot an$

Iată câteva reguli ce se referă la folosirea declarațiilor în limbajele de programare

- 1° Orice nume simbolic (identificator) folosit într-un program trebuie declarat¹ în primul rând ca tip și structură
- 2° Orice utilizare a unui identificator nu trebuie să contrazică declarația.
- 3° Un identificator nu poate fi redeclarat într-o zonă a programului în care este valabilă o primă declarație².

3 Pentru limbajul LPS declararea numelor simbolice o vom face printr-un tabel ce însoțește instrucțiunile fiecărui program. Acest tabel colectează toate numele simbolice folosite în program și le caracterizează din punct de vedere al domeniului din care acestea își iau valorile, prin atributele de structură și tip. Pentru această tabelul va avea trei coloane și anume, pentru nume, structură și tip. Dacă este cazul, putem adăuga și alte amănunte despre domeniul valorilor variabilelor, ele fiind utile proiectării și analizei algoritmului.

În figura 1.16 se dă reprezentarea completă a algoritmului 1 în LPS folosind și scrierea compactă a instrucțiunilor de bază de același tip, consecutive. S-a folosit prescurtarea v.s. pentru variabilă simplă.

<i>a</i>	v.s.	val. întregi
<i>b</i>	v.s.	val. întregi

citește *a*, *b*
 scrie *a*, *b*
 stop

Fig. 1.16

125. Probleme

1. Se dau două numere întregi pe banda de intrare în SPDS
 - a) Să se scrie algoritmul în LPS care înscrie pe banda de ieșire cele două numere în ordine inversă
 - b) Să se prezinte stările succesive ale SPDS în cursul execuției algoritmului pentru o pereche oarecare de valori
2. Să se scrie algoritmul care înscrie pe banda de ieșire valorile funcțiilor pentru valori date ale argumentelor:
 - a) $f(x) = e^{-x} \sin x$
 - b) $g(x, y) = \sqrt[3]{x^2 + y^2}$
 - c) $\phi(x, y) = r^5 \cos y^2$
 $\psi(x, y) = y^3 \ln(x^2 + 1)$
3. Se dă pe banda de intrare o valoare reală reprezentând valoarea unui unghi, dată în radiani. Să se scrie algoritmul în LPS care înscrie pe banda de ieșire valoarea unghiului în grade sexagesimale, minute și secunde. Se consideră $\pi = 3,14159$

¹ Declarația este de obicei explicită. În multe limbi se folosesc și declarațiile implicite sau prin lipsă. În asemenea cazuri lipsa unei declarații pentru un identificator asociază automat identificatorului un atribut cunoscut, micșor același.

² O asemenea restricție trebuie mai bine precizată pentru limbajele de programare cu structură de bloc. Datorită lipsei unor noțiuni necesare precizării ne mulțumim cu această formulare adecvată pentru limbaje cu FORTRAN și COBOL.

1.3. INSTRUCȚIUNILE DE CONTROL ALE LIMBAJULUI LPS

1.3.1. Instrucțiunea condițională

1. Prelucrările descrise de secvențele de instrucțiuni de bază sînt insuficiente. Să luăm un exemplu simplu :

Se dau două numere întregi pe banda de intrare a sistemului SPDS. Să se determine (prin înscriere pe banda de ieșire) cea mai mare valoare din cele două.

Bineînțeles, prima operație va fi să citim numerele și să le plasăm în două celule din memorie, să zicem x și y .

Cum va raționa un om care transmite instrucțiuni de bază calculatorului C pentru a putea rezolva problema ? Evident că întîi trebuie să determine din compararea celor două valori din memorie, care este cea mai mare valoare și de abia apoi să dea comanda de scriere. În cuvinte, raționamentul este următorul :

„dacă valoarea din x este mai mare decît valoarea din y atunci scrie x ,
altfel (adică dacă valoarea din x este mai mică sau egală cu valoarea lui y)
scrie y ”.

O formă generală pentru un asemenea raționament foarte frecvent întîlnit de noi în demonstrații sau calcule este :

„dacă *condiție* atunci
execută *ceva*
altfel
execută *altceva*”

Uneori întîlnim și o formă mai simplă :

„dacă *condiție* atunci
execută *ceva*”

În cazul că a doua alternativă nu execută nimic

Frecvența utilizării unui asemenea raționament ne obligă să introducem o operație de calcul care să l realizeze, cu atît mai mult cu cît raționamentul nu poate fi descris în termenii celor 4 instrucțiuni de bază. În termenii schemelor logice cele două forme ale raționamentului de mai sus se realizează cu schemele din figura I.17, *a*, *b*

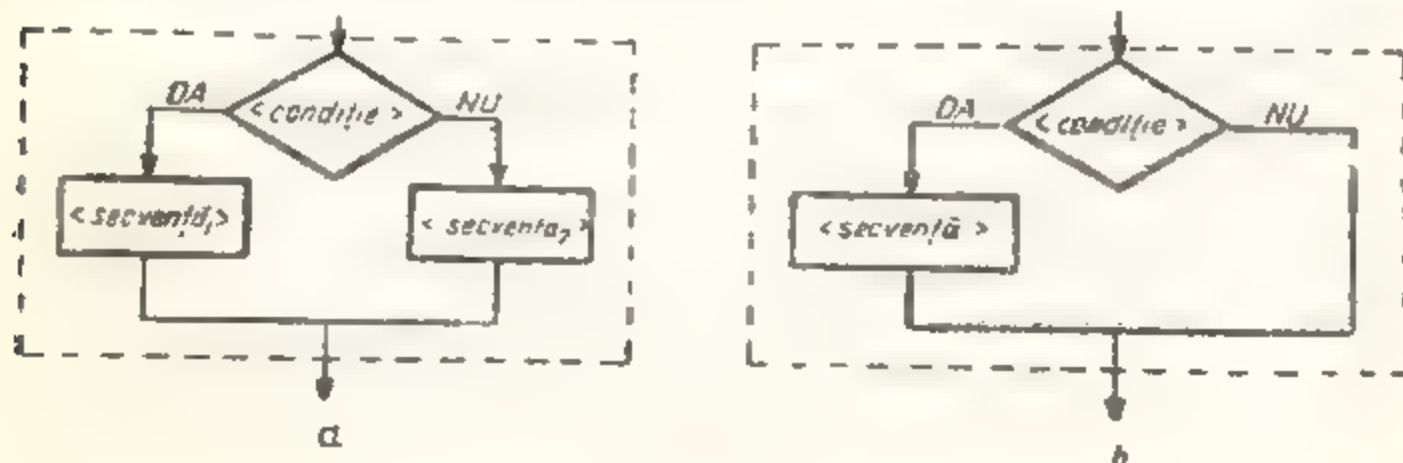
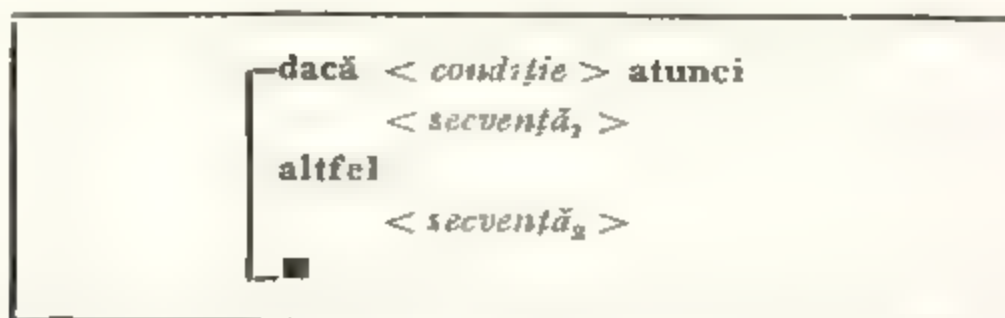
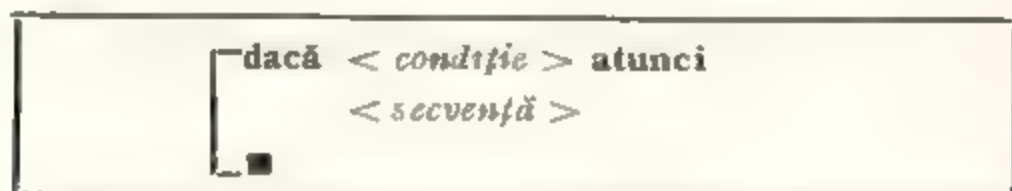


Fig. I.17

În LPS, prima variantă a raționamentului condițional o vom descrie cu ajutorul instrucțiunii condiționale sub forma :



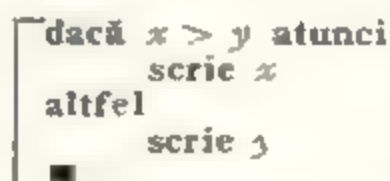
iar varianta a doua, cu o alternativă nulă ca efect, astfel :



În descrierile de mai sus < condiție > desemnează o expresie logică, deci o expresie care prin evaluare produce ca rezultat **adevărat** sau **fals** cum ar fi : $a > b$, $x = 0$, $alfa \leq 4 \vee beta = 5 < min$ în care ordinea de efectuare a operațiilor este cea cunoscută : întâi operațiile algebrice cu prioritățile respective, apoi operațiile relaționale și ultimele operațiile logice

< secvență >, < secvență₁ >, < secvență₂ > desemnează secvențe de instrucțiuni LPS de orice fel, inclusiv instrucțiuni condiționale.

Revenind la exemplul nostru, cu ajutorul instrucțiunii condiționale putem descrie operația prin care scriem pe BE cea mai mare valoare dintre cele două valori aflate în celulele x și y , astfel



2. După cum se vede, operația de decizie între două alternative a rezultat din necesitatea rezolvării unei probleme concrete, suficient de simple pentru a ne dovedi că realizarea de către calculator a unei asemenea operații este indispensabilă. *Forma în care am prezentat operația provine din modul în care raționăm și nu din modul în care calculatoarele reale o realizează.* De altfel, un calculator real nu poate realiza direct asemenea operații. Ele sînt simulate folosind operațiile elementare ale calculatorului și faptul că programul este memorat.

În figura I.18 este prezentat un exemplu al modului în care se simulează pe un calculator real instrucțiunea condițională.

Adresa din memoria program

Instrucțiuni din memoria program

Comentarii

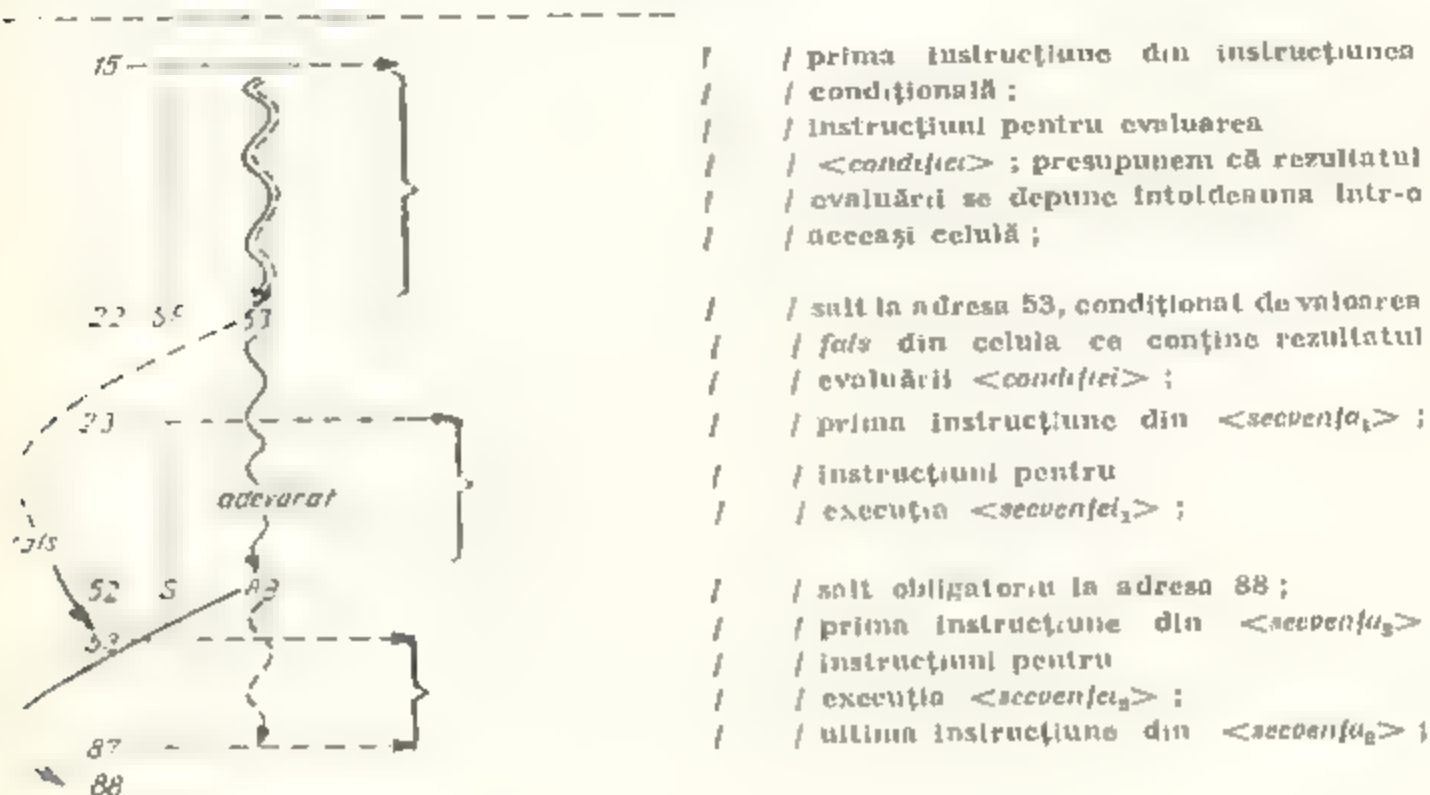


Fig. I.18

Instrucțiunea condițională simulată în limbajul calculatorului se întinde între adresele 15 și 87 ale memoriei programului. S-au pus în evidență instrucțiunea Sf 53 („salt la fals”) arată la adresa 22 din memoria programului și instrucțiunea S 88 („salt necondiționat”) arată la adresa 52. S-a rugat cu linie continuă cazul execuției primei alternative și cu linie întreruptă cazul execuției alternativei a doua.

Cu ajutorul C fiind imaginat de noi, ne putem permite să presupunem că el realizează decizia de a lua o decizie chiar dacă realizarea ei în cazul calculatoarelor reale constă de fapt din trei faze: a) evaluarea condiției; b) decizia de a urma o cale sau alta în execuție și c) execuția secvenței alese.

3. Din cele de mai sus rezultă caracterul deosebit al instrucțiunii condiționale față de instrucțiunile de bază. Din punct de vedere al efectului în instrucțiunile condiționale asupra unui calculator, se observă că operația de lansare de a schimba starea calculatorului doar prin modificările de sens în secvențele de instrucțiuni cuprinse în cadrul instrucțiunii, evaluarea condiției n-ducând la modificarea stării. Rolul operației este de a conduce execuția pe una din cele două căi în funcție de anumite relații existente între variabilele din program. Această decizie se ia *din timp*, la execuția programului.

Asemenea operație (instrucțiune), care stabilește o altă ordine de execuție a instrucțiunilor decât cea a seriilor acestora, se mai numește *operație (instrucțiune) de control* (al execuției).

Conținând la rândul ei alte instrucțiuni (datorită secvențelor incluse în sintaxa instrucțiunii), instrucțiunea condițională este un exemplu de instrucțiune structurată. Așa cum s-a mai arătat, instrucțiunile incluse pot fi chiar

instrucțiuni condiționale. Relația de includere este pusă în evidență prin scrierea decalată spre dreapta a secvențelor față de cuvintele **dacă** și **altfel**.

Algoritmul 2

Să se scrie algoritmul în LPS pentru rezolvarea ecuației de gradul I cu coeficienți reali.

Date inițiale : două valori reale. Le vom memora în cadrul programului în celulele a , b . Deci $a \in \mathbf{R}$, $b \in \mathbf{R}$.

Rezultate : o valoare reală care verifică ecuația cu coeficienții a și b . Să o notăm x . Deci : $x \in \mathbf{R}$

Între a , b și x avem relația $ax + b = 0$.

Algoritmul în LPS pare la prima vedere foarte simplu (fig. I.19).

a	v s.	val. reale
b	v s.	val. reale
x	v s.	val. reale

citește a , b

atribuie $x \leftarrow -b/a$

scrie x

stop

Fig. I.19

Să observăm că pentru $a = 0$, caz posibil deoarece $a \in \mathbf{R}$, ecuația devine

$$0 \cdot x + b = 0.$$

Relația este satisfăcută de orice $x \in \mathbf{R}$ dacă $b = 0$, dar niciodată satisfăcută de vreun $x \in \mathbf{R}$ dacă $b \neq 0$. De altfel, chiar împărțirea b/a trebuie să ne pună pe gânduri deoarece nu poate fi executată de calculator dacă $a = 0$ (împărțire cu zero).

Cazul $a \neq 0$ se poate rezolva ca în figura I.19. În celelalte două cazuri : $a = 0$ și $b \neq 0$ și respectiv $a = 0$ și $b = 0$, calculatorul nu poate realiza înscrierea rezultatului printr-o singură valoare deoarece avem fie nici o valoare, fie o infinitate de valori. În asemenea cazuri se obișnuiește să se tipărească pe BÎ un text (și de caractere) din care să reiasă situația respectivă.

Algoritmul 2 în noua versiune, îmbunătățită prin sesizarea tuturor cazurilor, este prezentat în figura I.20.

a	v s.	val. reale
b	v s.	val. reale
x	v s.	val. reale

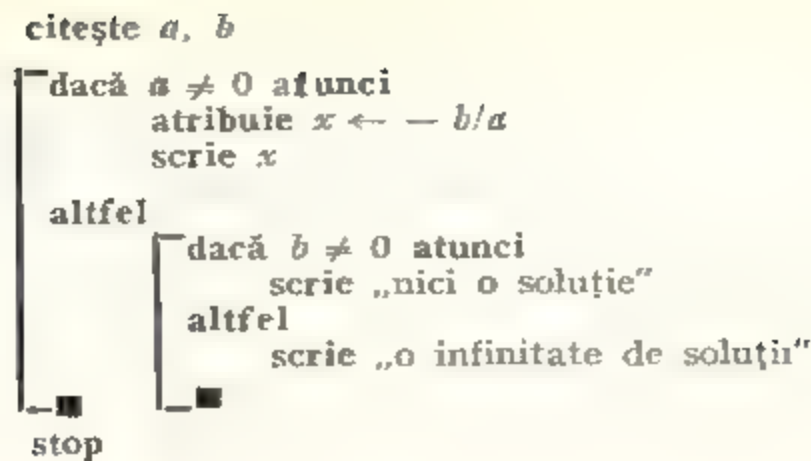


Fig. 1.20

Observație

În instrucțiunile de scriere s-a utilizat în loc de variabilă un text cuprins între ghilimele. Acest text poate fi privit ca o constantă de tip șir de caractere (așa cum 5 este constantă întreagă, 1.738 este constanta reală sau fals este constantă logică). Ea se înscrie în celula curentă a benzii BE exact sub forma șirului de caractere dintre ghilimele.

1.3.2. Instrucțiuni de ciclare cu condiție

1) Să ne reamintim metoda de determinare a cea mai mici divizor comun a două numere noulă și algoritmul lui Euclid și să o aplicăm pentru cazul particular al numerelor pozitive 308 și 110.

- 1° se împarte primul număr la al doilea determinând restul $308 = 2 \cdot 110 + 68 \rightarrow \text{rest} = 68$
- 2° se ia: al doilea din cele două numere și restul: 110 și 68
- 3° se împarte primul număr la al doilea determinând restul $110 = 1 \cdot 68 + 51 \rightarrow \text{rest} = 51$
- 4° se ia: al doilea din cele două numere și restul: 68 și 51
- 5° se împarte primul număr la al doilea determinând restul $68 = 1 \cdot 51 + 17 \rightarrow \text{rest} = 17$
- 6° se ia: al doilea din cele două numere și restul: 51 și 17
- 7° se împarte primul număr la al doilea determinând restul $51 = 3 \cdot 17 + 0 \rightarrow \text{rest} = 0$
- 8° se ia ca rezultat restul precedent (al doilea număr): 17.

Să observăm că:

1) Operațiile 1°, 3°, 5° și 7° sunt la fel și aceeași operație aplicată unor valori diferite. În acest fel primele 6 operații ale calculului constau din repetarea de 3 ori a aceluiași proces de operații.

11) Numărul de repetări nu era previzibil înaintea calculului, iar aplicând algoritmul pentru altă pereche de numere poate să rezulte un alt număr de repetări. De exemplu, pentru 1323 și 87 este necesară repetarea de 3 ori a operației de determinare a restului. Dece, modul de descriere folosit mai sus conduce la secvențe diferite de operații dacă este aplicat în aceeași manieră. Dacă ce dorim este să descriem într-un singur program (deci într-un text fix) repetarea de mai multe ori a aceleiași secvențe de operații, numărul de repetări fiind oricât de mare, dependent de îndeplinirea unei condiții.

Se constată relativ ușor că descrierea algoritmului lui Euclid în termeni instrucțiunilor învătate până acum, este imposibilă. Vom aborda o altă cale prin care se va impune ca necesară o operație nouă pentru calculatorul C și deci o nouă instrucțiune în IPS.

2) Să încercăm să expunem în cuvinte modul în care raționăm atunci când determinăm cu algoritmul lui Euclid c.m.m.d.c. a două numere.

Presupunem că am aplicat deja algoritmul lui Euclid pe câteva cazuri și ne putem permite generalizarea.

Se observă ușor că există un „grup de operații” care se repetă formînd un așa numit *ciclu* și anume

- determinarea restului celor două numere
 - pregătirea următoarelor două numere
- (A)

Deci am putea comanda calculatorului C execuția repetată a secvenței celor două operații prin :

„ciclează”

- determinarea restului celor două numere
 - pregătirea următoarelor două numere
- (B)

Numai că această repetare trebuie să se termine la un moment dat și anume cînd ultimul rest obținut este 0. Am putea să specificăm acest lucru în comanda execuției astfel

„ciclează”

- determinarea restului celor două numere
 - pregătirea următoarelor două numere
- (C)

„pînă cînd” restul este 0.

Cele două operații care se repetă pot fi descrise în LPS. Pentru aceasta vom plasa mai întîi valorile utilizate de aceste operații în memoria calculatorului. El sînt în primul rînd cele două numere, să le memorăm în celulele m și n . Apoi, restul să-l memorăm în celula r .

Cu acest „operațiunile din (C)” se pot descrie în LPS astfel

determinarea restului celor două numere

atribuie $r \leftarrow m - (m/n) \cdot n$

(D)

pregătirea următoarelor două numere

atribuie $m \leftarrow n$ & $n \leftarrow r$

Rămînă ca urmare a simbolului \vdash pentru împănare. Această operație de împănare are loc în \mathbb{Z} . De exemplu $17:5 = 3$, $7:9 = 0$ etc.

Rescriind forma (C) a programului conform descrierii în LPS din (D), se obține

„ciclează”

atribuie $r \leftarrow m - (m/n) \cdot n$

atribuie $m \leftarrow n$ & $n \leftarrow r$

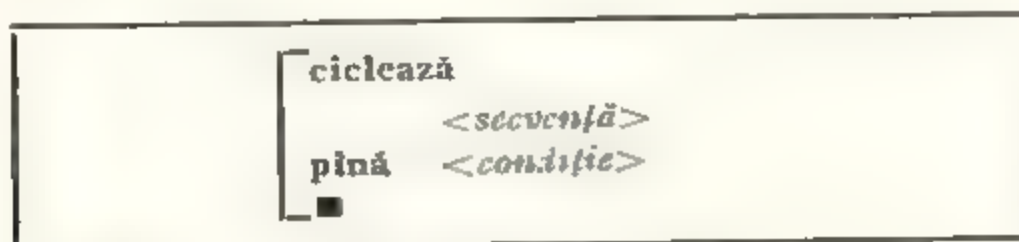
„pînă cînd” $r = 0$

În fapt, cunoaștem semnificația operației de terminare de cuvintele „ciclează”, „pînă cînd”. Ea nu se poate descrie cu ajutorul instrucțiunilor LPS învățate. Pe de altă parte operația se întâlnește frecvent în raționamentele noastre. De aceea vom introduce o nouă instrucțiune structurată în LPS, *instrucțiunea de ciclare cu test final*, care să determine repetarea sub controlul unei condiții a execuției unei secvențe de instrucțiuni. Este deci un alt exemplu de instrucțiune de control al execuției.

¹ În cazul în care folosim funcția de formă $\text{mod}(m, n)$ pentru restul modulo n al lui m , atribuirea se scrie astfel :

$$\text{atribuie } r \leftarrow \text{mod}(m, n)$$

Scrierea ei în LPS respectă forma de mai sus fixând cuvintele care îi determină semnificația :



Semnificația instrucțiunii astfel scrisă este echivalentă cu cea descrisă prin schema logică din figura I.21, a.

Secvența de instrucțiuni formează *corpul ciclului*.

Din execuția instrucțiunii de ciclare cu test final se observă o acumulare a efectelor execuțiilor succesive ale secvenței de instrucțiuni. Într-adevăr, executarea a n -a oară a secvenței de instrucțiuni ($n \geq 2$) se face plecând din starea SPDS rezultată în urma celei de a $(n - 1)$ -a execuții a aceleiași secvențe. Această stare a calculatorului trebuie să fie diferită de stările din care au început execuțiile precedente. În caz contrar ciclarea este infinită. Astfel, se poate observa că dacă la un moment dat în șirul de execuții ale secvenței realizat pînă în acel moment se identifică două execuții ale secvenței care lasă sistemul SPDS într-o aceeași stare, execuțiile vor continua la infinit. Condiția nu se va verifica niciodată. Același fenomen are loc dacă execuțiile secvenței nu schimbă nici una din variabilele din condiție.

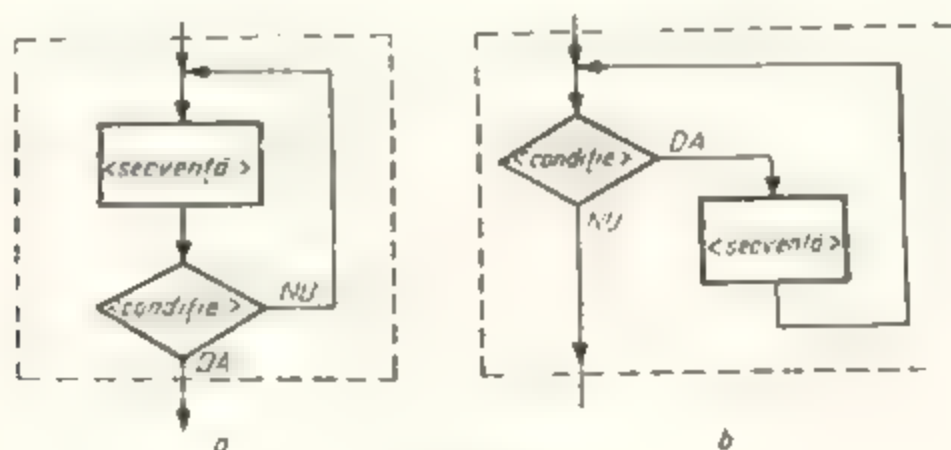
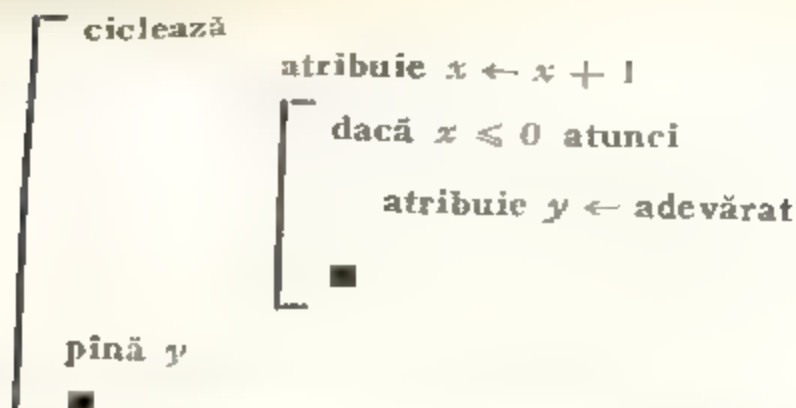


Fig. I.21

De exemplu :

x	v.s.	val. întregi
y	v.s.	val. logice

atribuie $x \leftarrow 0$ & $y \leftarrow \text{fals}$



Condiția de terminare a ciclării este formată din variabila y (al cărei tip este logic). Deci ciclarea se termină când y are valoarea *adevărată*. Acest lucru nu se va întâmpla însă niciodată, deoarece x își modifică valoarea fals doar dacă $x \leq 0$.

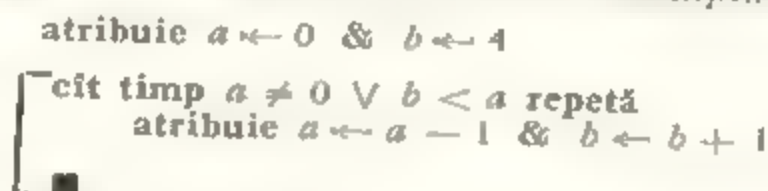
3. În unele prelucrări este utilă o altă instrucțiune de ciclare în care testul se face la început.

O vom scrie astfel :



Semnificația *instrucțiunii cu test la sfârșit* este echivalentă cu ceea ce s-a scris până la schema logică din figura 1.21, *b*.

Se observă că înălțurarea executului secvenței poate să nu aibă loc dacă, de la început, semnificația condiției este fals. De exemplu



Deoarece la început a este 0 și b este 4, condiția $a \neq 0 \vee b < a$ are valoarea fals, iar ciclarea nu are loc.

Algoritmul 3 (Algoritmul lui Euclid)

Să se descrie în LPS algoritmul pentru determinarea celui mai mare divizor comun a două numere întregi strict pozitive

Date inițiale : două numere întregi strict pozitive aflate pe BI

Rezultate : un număr întreg strict pozitiv care este c.m.m.d.c. al numerelor aflate pe BI.

Algoritmul în LPS pentru determinarea c.m.m.d.c. al celor două numere este prezentat în figura I.22. Celulele m și n folosesc pentru memorarea celor două numere aflate la un moment dat în discuție, iar celula r pentru memorarea restului.

m	v.s.	val. întregi pozitive
n	v.s.	val. întregi pozitive
r	v.s.	val. întregi pozitive
t	v.s.	val. întregi pozitive

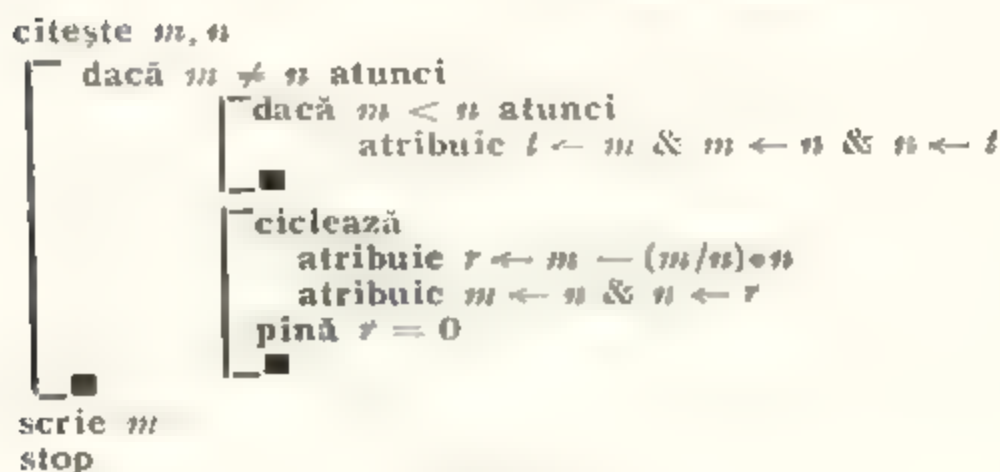


Fig. I.22

Din aplicarea algoritmului lui Euclid se știe că împărțind numărul mai mare la cel mai mic calculul are mai puține operații decât în cazul împărțirii numărului mic la cel mare. Este deci de dorit ca în m să se afle numărul mai mare și în n cel mai mic. De aceea, înainte de ciclare trebuie să ne asigurăm că acest lucru printr-o comparație a valorilor din m și n după care dacă în m valoarea este mai mică decât cea din n , vom schimba valorile între cele două celule. Această schimbare necesită trei atribuiri și o celulă suplimentară t în care se „salvează” una din valori. Succesiunea transferurilor de date între cele trei celule m , n și t este prezentată în figura I.23.

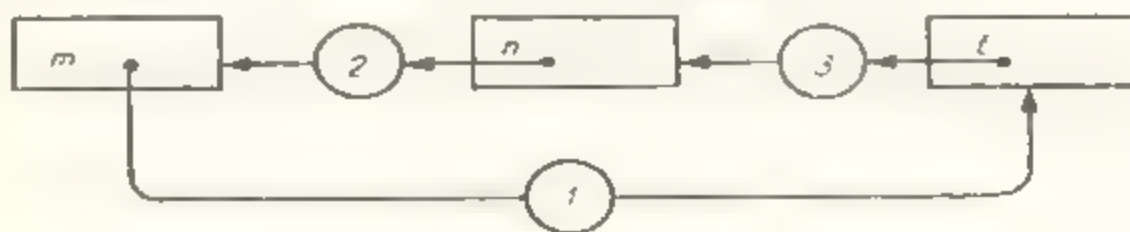


Fig. I.23

În fine, în figura I.22 se mai poate constata că s-au evitat calculele inutile pentru cazul $m = n$.

1.3.3 Utilizarea tablourilor în LPS

1. Ne propunem să scriem în LPS algoritmul pentru ordonarea de crescătoare a 5 numere reale aflate pe BI.

a) Fie, în ordine pe banda BI, următoarele 5 numere

$$315,12; 173,5; 82,9; -15; 33,2.$$

Evident, nu avem ce ordona, ele fiind ordonate, deoarece :

$$315,12 \geq 173,5 \geq 82,9 \geq -15 \geq 33,2$$

Deci le putem înscrie exact în această ordine pe banda BE, după ce l-am plasat temporar în 5 celule din memorie (*Drumul de la BI spre BE trece în mod obligatoriu prin memorie*).

b) Fie în ordine următoarele 5 numere pe banda BI

$$315,12; 173,5; -33,2; 82,9; -15$$

Ele nu mai pot fi transferate în această ordine pe BE, deoarece ordinul a d-
pe BI nu mai corespunde ordinii descrescătoare a celor 5 numere

$$-33,2 < 82,9.$$

Soluția este să le păstrăm decandată în memorie în vederea unei prelucrări având ca scop ordonarea valorilor. Odată ordonate le vom înscrie în ordine și respectiv pe banda BE. În figura 1.24 sunt prezentate cele 5 ce-

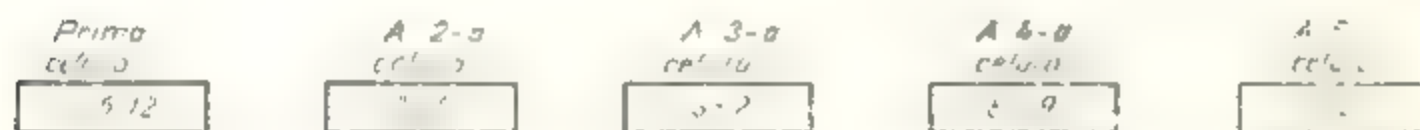


Fig. 1.24

lule din memorie ce conțin valorile citite. Celulele sunt și ele ordonate datorită unei numerotări: prima, a doua, ..., a cincea. Scopul nostru este să permutăm valorile în celule, astfel încât în prima celulă să avem cea mai mare valoare, în a doua valoarea următoare etc. În cazul din figură, înșurind de la stânga la dreapta celulele, o primă schimbare ar fi între celula a treia și a 4-a. Obținem șirul :

$$315,12; 173,5; 82,9; -33,2; -15.$$

După modificare șirul rămâne neordonat deoarece : $-33,2 < -15$

Din nou trebuie făcută o permutare de valori, de data această schimbând valorile între celulele a patra și a cincea, după care șirul este ordonat.

c) Fie în ordine următoarele 5 numere pe banda :

$$82,9; 33,2; 315,12; 173,5; -15$$

Aplicând compararea două câte două, de la stânga la dreapta, a numerelor memorate în această ordine descoperim că după $82,9 \geq 33,2$, avem $33,2 <$

$< 315,12$, deci locul lui 315,12 este în fața lui $-33,2$. Schimbăm prin urmare valorile între celulele a doua și a treia, ceea ce ne conduce la șirul :

82,9 ; 315,12 , $-33,2$, 173,5 -15 .

Dacă verificăm în continuare perechile, vedem că $-33,2$ trebuie să și schimbe locul cu 173,5 și apoi cu -15 , șirul devenind :

82,9 ; 315,12 , 173,5 ; -15 ; $-33,2$.

Rezultatul constă în plasarea celui mai mic număr în ultima celulă. Șirul nu este ordonat, deoa-ecce, reluând compararea, constatăm că $82,9 < 315,12$, deci trebuie schimbate valorile primelor două celule ș.a.m.d.

Rezumând, putem spune că ordonarea are loc inspectând valorile aflate în perechi de celule consecutive în încercarea de a le ordona corect. În prima celulă valoarea mai mare, în a doua celulă valoarea mai mică. Dacă valorile lui sînt deja ordonate, trecem la următoarea pereche de celule, dacă nu, permutăm cele două valori. Ordonarea se termină cînd, parcurgînd încă o dată șirul celulelor, constatăm că nu mai este necesară nici o permutare.

2. Să observăm că în ceea de mai sus am considerat de la început celulele (nu valorile !) ordonate. Dacă am convenit de la început aceasta este celula care în final va conține cel mai mare număr, ea este prima, aceasta este celula ce va conține în final următorul număr, ea este a doua ș.a.m.d. Să presupunem că le-am și denumit în vederea scrierii programului ; în ordine sînt a, b, c, d, e . Atunci o trecere prin șirul valorilor va fi înscrisă în LPS astfel :

```

| dacă  $a < b$  atunci
|   atribuie  $t \leftarrow a \ \& \ a \leftarrow b \ \& \ b \leftarrow t$ 
| ■
|
| dacă  $b < c$  atunci
|   atribuie  $t \leftarrow b \ \& \ b \leftarrow c \ \& \ c \leftarrow t$ 
| ■
|
| dacă  $c < d$  atunci
|   atribuie  $t \leftarrow c \ \& \ c \leftarrow d \ \& \ d \leftarrow t$ 
| ■
|
| dacă  $d < e$  atunci
|   atribuie  $t \leftarrow d \ \& \ d \leftarrow e \ \& \ e \leftarrow t$ 
| ■

```

Se constată că sînt necesare 4 instrucțiuni condiționale pentru a trece în revistă cele 5 numere. Cu 100 de numere ar fi necesare 99 instrucțiuni condiționale, iar cele 100 celule distincte le am denumit probabil : $a, b, \dots, z, ab, bc, \dots, t, \dots$. Dar dacă ar fi de ordonat 10 000 de numere, cum am denumi celulele și cîte instrucțiuni condiționale ar fi necesare ? E clar că pentru a rezolva metoda de ordonare a unei mulțimi oarecare de numere trebuie în cercat altfel.

3. Vom pleca de la observația că cele 4 instrucțiuni condiționale de mai sus pot fi rezolvate la fel o pereche de celule, pereche care însă se schimbă de la un caz la altul. Într-adevăr, cu excepția numelor celulelor, cele 4 instrucțiuni sînt identice. Nu s-ar putea să scriem prelucrarea realizată o singură dată, dar s-o executăm de patru ori schimbînd la fiecare execuție, printr-un anumit

procedeu, numele celulelor asupra cărora lucrăm? Pentru aceasta vom considera cele 5 celule ca un ansamblu caruia îi dăm un nume, iar în cadrul ansamblului vom numerota celulele de la 1 la 5 în vederea deosebirii lor (fig. 1.25).

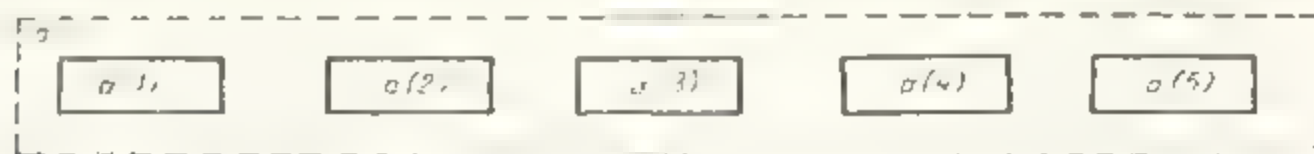
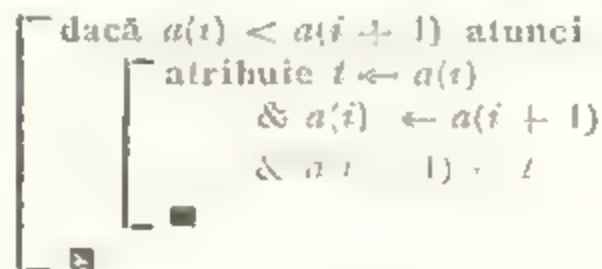


Fig. 1.25

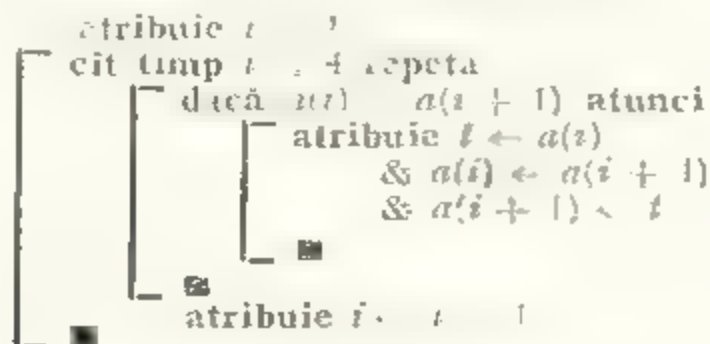
Reamintim că un asemenea ansamblu formează, conform celor discutate în 1.2.4, un *tablou*, iar o celulă din ansamblu se numește *element de tablou*. Pentru a identifica tabloul folosim un nume ca și în cazul variabilelor, iar pentru a identifica un element de tablou folosim numele celulei din care face parte, precum și numărul de ordine. Acest număr de ordine poartă numele de *indice*. În descrierea algoritmului în LPS, în tabloul asociat algoritmului specificăm faptul că este vorba de un tablou, numărul dimensiunilor tabloului și domeniul valorilor indicilor pentru fiecare dimensiune. În referințe la un element de tablou vom folosi notația indicelui între paranteze. Pentru $a(3)$ este vorba de al treilea element al unui tablou numit a , care, evident, va avea cel puțin 3 elemente. Un nou valoare de indice poate fi memorată într-o altă celulă din memorie. De exemplu $tab(i)$ unde i este o celulă din memorie ce conține o valoare întreaga pozitivă. Evident, valoarea lui i trebuie să fie mai mică sau egală cu numărul de celule al tabloului tab .

În fine, indicele poate fi dat și de o expresie oarecare ce are ca valoare un număr întreg pozitiv. De exemplu $a(i + 1)$ unde i și 1 sînt celule ce conțin valori întregi. Deoarece o asemenea notatie identică, ca și în cazul unei variabile, o celulă din memorie vom face distincție între *variabilă* și *valoare* (notăm pentru un element de tablou și *element de tablou* folosind pînă acum pentru o celulă singulară din memorie).

Revenim la căutarea minimelor vom scrie algoritmul condițional care se repetă, sub forma:



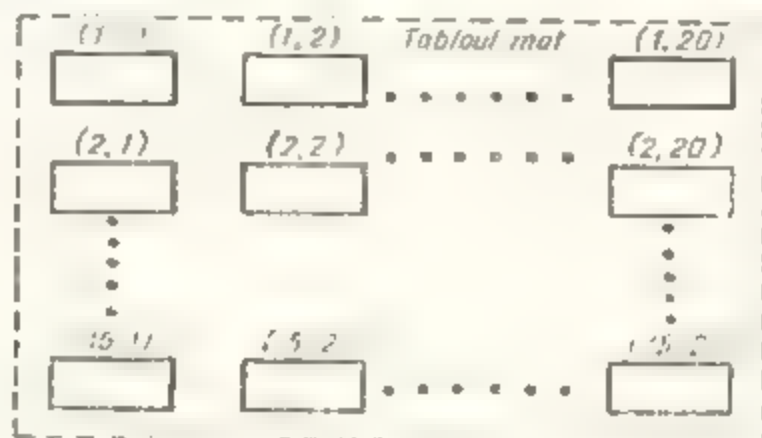
și vom obliga pe i să ia valori de la 1 la $n - 1$ astfel:



4 Conform celor prezentate în I 2 4 un tablou ca cel de mai sus este și milar unu vector . mulțime de valori ordonate după un singur criteriu . Putem extinde aceste considerații și pentru matrice, deci mulțime de valori ordonate după două criterii .

Să ne închipuim o zonă din memoria M cum este cea din figura I.26, să-i dăm un nume , *mat* și să localizăm o celulă din această zonă prin doi indici : indicele liniei și, respectiv, indicele coloanei . Astfel, *mat*(2,5) este numele celulei din tabloul *mat* aflată pe linia a 2 a și coloana a 5 a .

Fig. I.26



Valorile aflate în celulele zonei *mat* sînt toate de același tip și formează de fapt un tablou de valori (— lași cu cel din exemplul din I 2 4), ce poate fi asimilat cu o matrice .

Observație Plasarea în memorie a celulelor în cazul calculatoarelor reale nu corespunde exact situației din figura I.26, deoarece memoria acestora este lineară . De aceea descrierea cu mai mulți indici a elementelor de tablou trezume linearizată de către compilatorul limbajului de programare .

În fine, putem să extindem considerat de la mulțime de valori ordonate după mai multe criterii , ca și la zone de memorie , tablouri în care celulele sînt localizate folosind tot un număr de doi indici, deci cu mai mult de două dimensiuni . De exemplu, *to* = (3, 5, 8) este o celulă dintr-un tablou în care accesul se face pe baza a 2 indici : indicele de linie, cel de coloană și cel de plan (trebuie totuși în această zonă în spațiu).

Algoritmul 4 (ordonarea numerelor)

Să se scrie algoritmul în LPS pentru înscrierea pe BE în ordine descrescătoare a unui șir de valori reale aflate pe banile BI . Numărul acestor valori (număr întreg pozitiv) se află în prima celulă a benzii BI . El este mai mic decât 100 .

Dat : un șir de valori reale pozitive și apoi un șir de numere reale de lungime egală cu numărul întreg .

Rezultat : un șir de numere reale, același cu cele citite, dar în ordine descrescătoare a valorilor .

Procedeul de ordonare a fost prezentat mai sus pentru cazul a 5 numere . Vom „construi” descrierea algoritmului în LPS pentru un caz mai general în acționând să descriem pas cu pas ceea ce trebuie făcut pentru a realiza cu ajutorul C ordonarea cerută . Metoda de construire a algoritmului este deosebit de importantă , de aceea recomandăm abordarea ei cu atenție .

ntorcându-ne la (2)' să observăm că atenția acordată exclusiv lui (2.1)' a făcut să fie dificilă descrierea în LPS a ceea ce înseamnă (2.2). De aceea va trebui să revenim în (2.1)' și (2.1.1)', să le modificăm, pentru a putea descrie în LPS operația (2.2). Să acordăm mai multă atenție acestei reveniri pentru a evita repetarea greșelii pe viitor.

Este clar că (2.2) va folosi informația pusă la dispoziția sa de către (2.1). Deci există una sau mai multe celule din memorie ce vor găzdui aceste informații transmițându-le de la operația (2.1) la operația (2.2). Informația necesară în (2.2) este răspunsul la întrebarea: "lu o trecere prin șirul de valori și a realizat vreo permutare a valorilor sau nu?". În primul caz, șirul nu este ordonat; în al doilea caz el este ordonat. Informația este cu două valori: "da" sau "nu", "0" sau "1", **adevărat** sau **fals**. Să considerăm ultima codificare și o celulă *sem* care "semnalizează" prin **adevărat** dacă nu s-a făcut nici o permutare și prin **fals** dacă a existat măcar o permutare. Celula este actualizată (modificată) de către (2.1) și testată de (2.2). Vom scrie acest lucru astfel:

```
(2)'' [ ciclează
        (2.1) * inspectează și permută într-o trecere șirul (n, a;
              a, sem)
        pînă (2.2) * șirul este ordonat (sem);
```

În (2)'' apar în plus față de (2)' pentru fiecare enunț nestandard niște argumente între paranteze. Argumentele indică sub forma unor liste de nume simbolice celulele din memorie care afectează sau care sunt afectate de operația respectivă. Prima categorie de celule formează lista de intrare, iar a doua, lista de ieșire. Cele două liste sînt separate prin \leftarrow . După cum se vede, (2.1) are nevoie de celulele *n* și *a* (toate celulele tabloului), informațiile afilate în ele determinînd acțiunea, efectul lui (2.1). În același timp (2.1) poate modifica conținutul celulelor *a* și *sem*. Spunem că cele două liste formează „interfața” operației respective cu celelalte operații. Astfel, *sem* este celula ce formează „interfața” dintre (2.1) și (2.2) și am văzut că am fost obligați s-o folosim totuși pentru transferul informațiilor de la (2.1) la (2.2).

Specificarea variabilelor interfeței unei operații nestandard cu celelalte operații din program este utilă în faza de proiectare a programelor, mai ales în cazul programelor complexe, de mari dimensiuni.

Să reluăm detalierea operației (2) adăugînd interfețele operațiilor nestandard.

```
(2.1)'' atribuire sem  $\leftarrow$  adevărat
        atribuire i  $\leftarrow$  1
        [ cît timp i  $\leq$  n - 1 repetă
          (2.1.1) * compară două celule consecutive (i, a, a, sem)
          atribuire i  $\leftarrow$  i + 1
```

Să observăm că **atribuire *sem* \leftarrow adevărat** echivalează cu asumarea unei ipoteze: șirul este ordonat. Urmează ca (2.1.1) să ne contrazică sau nu prin efectul execuției ei asupra lui *sem*.

```

(2.1.1)''
┌─ dacă  $a(i) < a(i + 1)$  atunci
│   ┌─ atribuie  $t \leftarrow a(i)$ 
│   │   &  $a(i) \leftarrow a(i + 1)$ 
│   │   &  $a(i + 1) \leftarrow t$ 
│   └─ ■
└─ atribuie  $sem \leftarrow fals$ 
  └─ ■

```

Iată cum (2.1.1)'' dă valoare complet rolul lui *sem* — acela de „semafor“ — că dă „liber“ sau nu operației de transcriere a valorilor pe BI. Într-adevăr, dacă $a(i) < a(i + 1)$ urmează o schimbare a valorilor acestor celule în acest $i, i + 1$ și o atribuie a valori **fals** lui *sem* — ceea ce de altfel urmăream. Celui *sem*, după cum se vede din (2.1)'', nu mai capătă valoare **adevărat** în cursul parcurgerii curente a șirului, ci cel mult i se mai confirmă de câteva ori valoarea **fals**.

Exercițiu: Verificați acest lucru executând secvența (2.1)'' pentru șirul 1,7 ; -2,25 ; 10,3.

Ne rânim să scriem cine este în nouă condiția (2.2), ceea ce este foarte simplu :

(2.2)' *sem*

Revenind la prima formă a programului, să o rescriem și pe aceasta scriind interfetele :

```

(0)'          (1) • citirea valorilor de pe BI ( $n, a$ )1
              (2) • ordonarea valorilor în memorie ( $n, a ; a$ )
              (3) • înscirerea valorilor ordonate pe BI ( $n, a$ )
stop

```

Nă vom îndrepta atenția spre operațiile (1) și (3). (1) înseamnă citirea unui număr de valori de pe BI, număr pe care nu îl cunoaștem acum, când scriem programul, deci lista identităților din instr. (1) și (3) citire are o lungime necunoscută ce depinde de prima valoare citită. La plus, valorile citite se vor depune în celule aparținând tabloului *a*. Dacă ar fi să scriem numele acestor celule, ar trebui să facem o listă, astfel :

$a(1), a(2), a(3), \dots, a(n)$

unde *n* este numele unei celule din memorie în care s-a aflat numărul de valori de pe BI.

Deoarece specificația nu este precisă (formularea că „*n* nu este admisibil“), va trebui să specificăm într-o manieră riguroasă modul în care se determină valorile consecutive ale indicii. Vom introduce notația $(a(i), i = 1, n)$ pentru lista de nume sus. Se observă că între paranteze se găsește termenul general $a(i)$ al tabloului *a* și expresia $i = 1, n$, care, prin convenție, ne arată că *n* ia valori celule i de la 1 pînă la valoarea din *n* — fiind mereu cu 1 valoare precedentă.

Deci putem detalia operația (1) astfel :

```

(1)'          citește n
              citește ( $a(i), i = 1, n$ )

```

¹ Se observă lipsa porțiunii de scriere — lipsa listei de citire — operația (1) dar în operația (3) lipsa listei de citire.

La fel pentru operația (3):

(3)' scrie $(a(i), i = 1, n)$

Înlocuind succesiv formele finale, în LPS, ale operațiilor nă standard și adău-
gînd declarațiile atributelor variabilelor utilizate obținem algoritmul din
figura I.27.

De multe ori este util să menționăm, dacă este posibil, un domeniu mai
restrîns decît mulțimile \mathbf{Z} , \mathbf{R} , pentru variabilele utilizate în program. Acest
lucru l-am făcut în tabelul din figura I.27, pentru variabila n . Domeniul spe-
cificat prin expresia $[1, 100]$ este cel al tuturor valorilor întregi între 1 și 100.

n	var. simplă	valori întregi $[1, 100]$
a	tablou 1 dimens. (100)	valori reale
i	var. simplă	valori întregi
t	var. simplă	valori reale
sem	var. simplă	valori logice

```

citește  $n$ 
citește  $(a(i), i = 1, n)$ 
  ciclează
    atribuie  $sem \leftarrow$  adevărat
    atribuie  $i \leftarrow 1$ 
    cît timp  $i \leq n - 1$  repetă
      dacă  $a(i) < a(i + 1)$  atunci
        atribuie  $t \leftarrow a(i) \& a(i) \leftarrow a(i + 1) \& a(i + 1) \leftarrow t$ 
        atribuie  $sem \leftarrow$  fals
      atribuie  $i \leftarrow i + 1$ 
    pînă  $sem$ 
  scrie  $(a(i), i = 1, n)$ 
  stop
  
```

Fig. I.27

O astfel de construire pas cu pas a algoritmului unei probleme prin deta-
lierca componentelor nă standard este o modalitate tot mai frecvent utilizată în
proiectarea sistematică a programelor. Se numește *proiectare descendentă*
pas cu pas a programului.

1.3.4. Instrucțiunea de ciclare cu contor

1. Să reluăm secvența (2.1)'' din exemplul precedent, de ordonare a unor
valori reale, și să ne amintim rolul variabilei simple i în prelucrarea consi-
derată. În afara de faptul că ia valori întregi, să mai observăm că aceste valori

se află între 1 și valoarea din n . Într-adevar, i ia valoarea 1 înainte de a intra în ciclul ca test inițial, iar în interiorul ciclului, la sfârșitul acestuia, valoarea din i crește cu 1 până în momentul în care $i > n - 1$ (adică $i = n$). În acest moment repetarea execuției instrucțiunilor din ciclu se oprește. Putem spune că i numără astfel executule secvenței din ciclu sau le *contori*ază. De aici denumirea de *contor* dată unei asemenea variabile.

Prelucrările cu contor sînt frecvente în aplicații și de aceea se pot întâlni în multe limbaje de programare instrucțiuni speciale pentru descrierea lor. Vom introduce și noi o asemenea instrucțiune în LPS, observînd totuși că ea poate fi „simulată” cu ajutorul instrucțiunilor de ciclare cu test inițial sau final.

Instrucțiunea de ciclare cu contor se scrie în LPS astfel:

[pentru $\langle var \rangle = \langle init \rangle$ - $\langle fin \rangle$, $\langle pas \rangle$ execută
 $\langle secvență \rangle$
]

unde $\langle var \rangle$ este contorul, o variabilă simplă de tip întreg, iar $\langle init \rangle$, $\langle fin \rangle$, $\langle pas \rangle$ sînt numere sau variabile simple întregi. Dacă $\langle pas \rangle$ este 1, el poate lipsi împreună cu virgula ce l precede.

Efectul acestei instrucțiuni este dat de schema logică din figura 1.28.

2. Să observăm o deosebire importantă între ciclarea cu contor și cea cu condiție. Chiar dacă în momentul în care se scrie programul se poate să nu se știe de cîte ori va fi executată secvența, acest lucru devine cunoscut

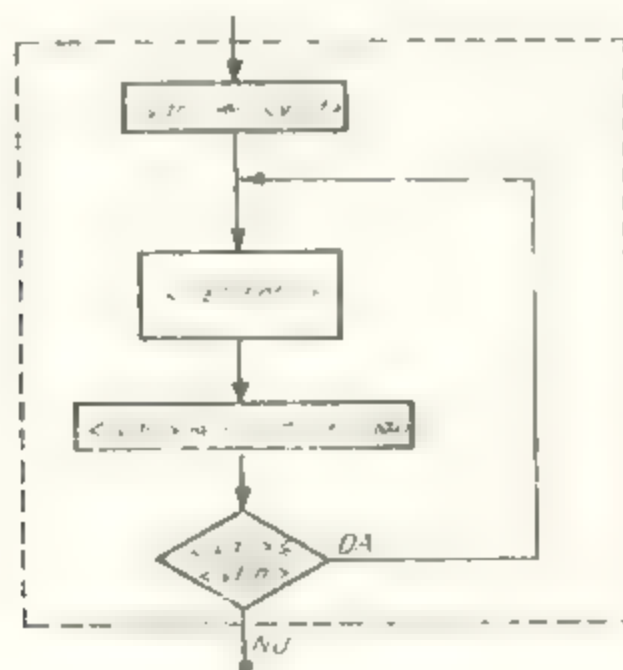


Fig. 1.28

atunci cînd se începe execuția înlănțuită a secvenței rezultînd din calculul: $(fin - init) / pas + 1$. În ciclarea cu condiție sfîrșitul ciclării rezultă de obicei ca o consecință a efectului înlănțuirii respective. În aceste condiții nu se poate spune nici măcar la începutul execuției operației de ciclare cu condiție cînd se va termina ea.

Reluând programul de ordonare din figura I.27 și refacându-l pentru a folosi instrucțiunea de ciclare cu contor, rezultă programul din figura I.29.

n	var. simplă	valori întregi [1, 100]
a	tablou 1 dimens. (100)	valori reale
i	var. simplă (contor)	valori întregi
t	var. simplă	valori reale
sem	var. simplă	valori logice
m	var. simplă	valori întregi

```

citește  $n$ 
citește ( $a(i)$ ,  $i = 1, n$ )
┌
│   ciclează
│       atribuie  $sem \leftarrow$  adevărat
│       atribuie  $m \leftarrow n - 1$ 
│
│   ┌   pentru  $i = 1, m$  execută
│   │   ┌   dacă  $a(i) < a(i + 1)$  atunci
│   │   │   atribuie  $t \leftarrow a(i)$  &  $a(i) \leftarrow a(i + 1)$  &  $a(i + 1) \leftarrow t$ 
│   │   │   atribuie  $sem \leftarrow$  fals
│   │   └   ──
│   └   ──
│   pînă  $sem$ 
└   ──
scrie ( $a(i)$ ,  $i = 1, n$ )
stop

```

Fig. I.29

De remarcat artificul care asigură o variabilă simplă ca „ a/m ” în instrucțiunea de ciclare cu contor — s-a introdus o variabilă m care înaintea instrucțiunii de ciclare a fost inițializată cu valoarea unei expresii ce depinde de limita superioară a valorii contorului.

13.5. Probleme

- Se dau două numere întregi
 - Să se scrie algoritmul în LPS ce inscrie pe bandă de testare cele două numere în ordine descrescătoare.
 - Să se prezinte stările succesive ale SPDS în cursul execuției algoritmului.
- Idem pentru trei numere întregi.
- Să se scrie algoritmul în LPS pentru rezolvarea ecuației de gradul II cu coeficienți reali. *Indicație:* Rădăcinile complexe conjugate se pot inscrie ca două valori reale — partea reală și partea imaginară. Pentru a nu confunda acest caz cu cel al rădăcinilor reale distincte este bine ca pe banda de testare să se tipărească și un text care să informeze vizitatorul asupra situației în care s-a terminat execuția.

4. Să se scrie algoritmul în LPS pentru a determina valoarea funcției date pe intervale :

$$f(x) = \begin{cases} 1 - x & \text{dacă } x < -1, \\ \sqrt{1 - x^2} & \text{dacă } -1 \leq x \leq 1, \\ x - 1 & \text{dacă } x > 1. \end{cases}$$

5. Se dau pe banda de intrare două date calendarice, fiecare formată din trei valori întregi, strict pozitive reprezentând ziua, luna și respectiv anul. Prima dată este data în care se execută programul. A doua reprezintă data nașterii unei persoane. Să se scrie programul în LPS care înregistrează pe banda de ieșire vârsta în ani a persoanei respective. Dacă aceasta își serbează ziua de naștere în chiar ziua executării programului, se consideră că are o vârstă cu un an mai mare decât cea din ziua precedentă.

6. Fie algoritmul din figura 1.30

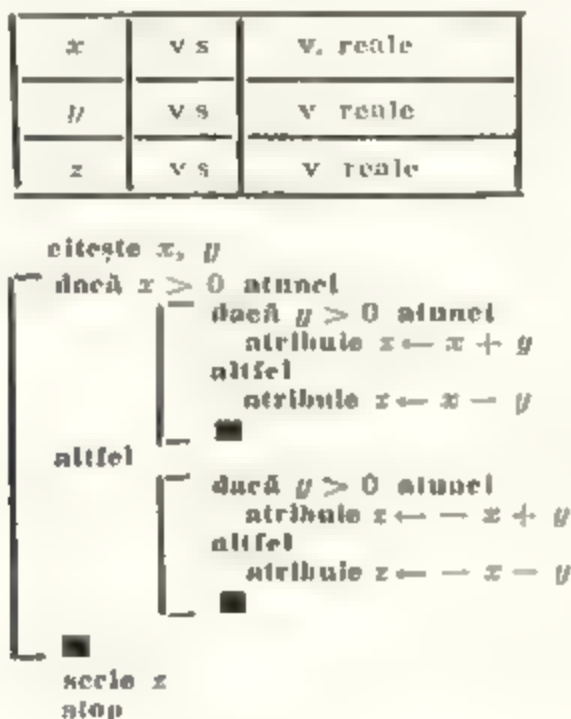


Fig. 1.30

Să se arate că algoritmul verifică condiția $z = |x| + |y|$

7. Fie algoritmul din figura 1.31 reprezentat printr-o schemă logică în care m și n sînt variabile întregi strict pozitive.

a) Să se verifice că algoritmul este o variantă a algoritmului lui Euclid

b) Să se scrie algoritmul în LPS care descrie aceeași prelucrare

8. Să se scrie algoritmul în LPS pentru calculul factorialului unui număr întreg pozitiv dat

9. Fie ecuația $x^2 - 1 = 0$. Să se scrie algoritmul în LPS pentru determinarea, cu o eroare absolută maxim admisibilă dată, a unei soluții aproximative a ecuației. Se va utiliza metoda înjumătățirii intervalului

Indicații

În intervalul $[0, 1]$ se găsește o singură soluție exactă x_0 . Într-adevăr, fie $f(x) = x e^x - 1$, atunci $f(0) = -1 < 0$ și $f(1) = e - 1 > 0$, iar în acest interval $f(x)$ este continuă și monoton crescătoare :

Să notăm cu x_0 soluția exactă, x_n cea aproximativă și ε eroarea absolută maxim admisibilă. Fie $[a, b]$ un interval în care știm că se găsește soluția exactă, atunci :

$$(\forall x_0)(x_n \in [a, b])(a - b) < \varepsilon \Rightarrow |x_0 - x_n| < \varepsilon$$

Deci este suficient să găsim un interval $[a, b]$ astfel încît $|a - b| < \varepsilon$ pentru ca luînd orice punct x_n din acest interval să fim asigurați că $|x_0 - x_n| < \varepsilon$.

- Dacă $[a, b]$ este un interval în care se găsește soluția exactă atunci :

sau $f(a) \leq 0$ și $f(b) > 0$, sau $f(a) < 0$ și $f(b) \geq 0$.

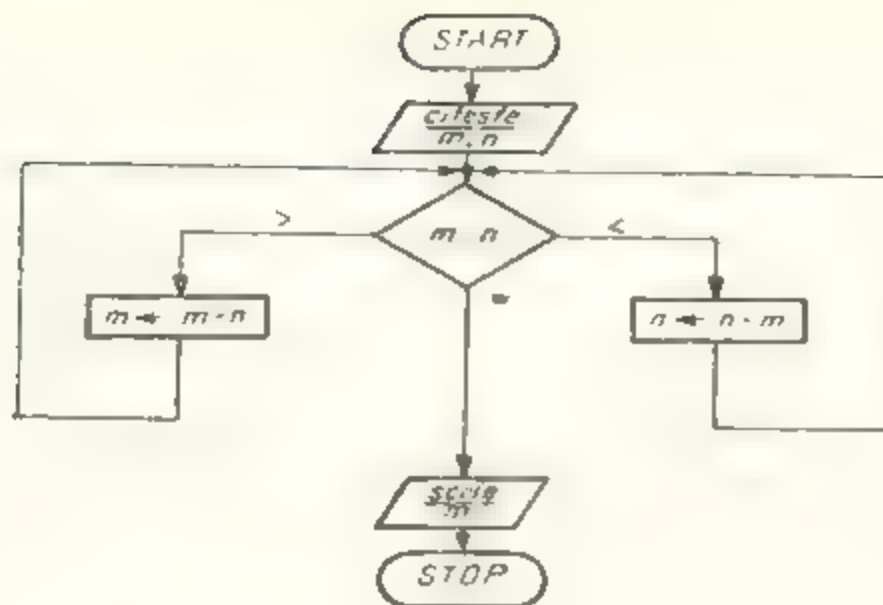


Fig. I.31

10. Să se scrie algoritmul în I.P.S. pentru calculul rădăcinii de un ordin întreg pozitiv dat dintr-un număr real de asemenea dat. Rădăcina se calculează cu o eroare absolută dată (valoare reală).

Indicație

Se poate folosi metoda lui Newton de calcul a rădăcinii de ordinul p dintr-o valoare a (adică $\sqrt[p]{a}$). Conform acestei metode, șirul cu termenul general $x_n = \frac{1}{p} \left[(p-1)x_{n-1} + \frac{a}{(x_{n-1})^{p-1}} \right]$ converge spre $\sqrt[p]{a}$. Numărul rădăcinilor pe care le putem obține folosind acest șir depinde de semnul lui a și de paritatea lui p astfel:

	$a > 0$	$a < 0$
p par	2 răd. $a^{1/p}, -a^{1/p}$	nici o răd.
p impar	1 răd. $a^{1/p}$	1 răd. $- a ^{1/p}$

☞ Alegerea lui x_0 are de asemenea importanță pentru determinarea rădăcinii astfel:

	p par, $a > 0$	p impar, $a > 0$	p impar, $a < 0$
$x_0 < 0$	$-a^{1/p}$	$a^{1/p*}$	$-(a)^{1/p}$
$x_0 > 0$	$a^{1/p}$	$a^{1/p}$	$-(a)^{1/p*}$

* În acest caz nici un termen x_n nu trebuie să fie nul

Evident: $x_0 = x_1 \Rightarrow x_0 = x_1 = \dots = x_n = \dots$

x_0 trebuie să fie de asemenea dat.

Eroarea absolută, z-o notăm ε , servește la oprirea calculului. Metoda lui Newton ne dă posibilitatea să evaluăm eroarea $|x_n - \sqrt[p]{a}|$ majorând-o astfel:

$$|x_n - \sqrt[p]{a}| < |x_1| \prod_{i=1}^{n-1} \left[\left(\frac{p-1}{p} \right) \left(1 - \frac{a}{(x_i)^p} \right) \right].$$

Deci:

$$|x_n - \sqrt[p]{a}| < |x_1| \prod_{i=1}^{n-1} \left[\left(\frac{p-1}{p} \right) \left(1 - \frac{a}{(x_i)^p} \right) \right] < \varepsilon \Rightarrow |x_n - \sqrt[p]{a}| < \varepsilon$$

11. a) Să se scrie algoritmul în LPS care extrage dintr-un șir de valori reale dat valoarea cea mai mare și o înscrie pe bandă de ieșire. Șirul este dat pe bandă de intrare precedat de o valoare întregă pozitivă reprezentând numărul de valori reale din șir (mai mic decât 1000).

b) Încercați să rezolvați această problemă folosind un număr cel mai mic de celule ale memoriei.

Indicație. La un moment dat nu sînt necesare în memorie toate valorile reale.

12. Se dau două puncte într-un spațiu tridimensional prin coordonatele lor (valori reale). Să se scrie în LPS algoritmul care rezolvă următoarele probleme:

a) Să se afle distanța dintre cele două puncte;

b) Știind că cele două puncte sînt vîrfurile a doi vectori cu originea în centrul axelor de coordonate, să se afle produsul scalar al celor doi vectori, unghiul dintre cei doi vectori și coordonatele vîrfului produsului vectorial al celor doi vectori;

c) Să se generalizeze problemele a) și b) pentru un spațiu n -dimensional (cu n dat).

13. Se dă banda de intrare din problema 11 (un șir de valori reale precedate de numărul lor). Să se scrie algoritmul în LPS care înscrie pe bandă de ieșire:

a) suma valorilor din șir,

b) produsul valorilor din șir,

c) produsul valorilor nenule din șir,

d) suma pătratelor valorilor din șir,

e) media aritmetică a valorilor din șir,

f) media geometrică a valorilor pozitive din șir,

g) media armonică a valorilor nenule din șir,

h) dispersia valorilor din șir și abaterea medie pătratică.

Indicație. Dacă x_1, x_2, \dots, x_n sînt valorile din șir, dispersia D și abaterea medie pătratică σ sînt date de relațiile

$$D = \frac{1}{n} \sum_{i=1}^n (x_i - m)^2, \quad \sigma = \sqrt{D}, \quad \text{unde } m = \frac{1}{n} \sum_{i=1}^n x_i \text{ este media aritmetică a valorilor}$$

14*. Variante ale algoritmului de ordonare.

a) Analizăm algoritmul de ordonare din figura 1.27 bazat pe metoda „bule” sau a „propietății” și observăm că după a j -a parcurgere a șirului de numere, ultimele j valori sînt deja ordonate. Deci luăm la supraîncălzire pentru creșterea lui j peste f , $n = (j + 1)$ în loc de $n - 1$ (unde j numără parcurgerile șirului).

Reverșim algoritmul care să includă această optimizare.

b) Altfel, datorită unei eventuale ordonări parțiale a șirului încă de la începutul execuției s-ar putea ca nu mult de j valori de la sfîrșitul șirului să fie ordonate. Dar cum putem ști, cite sînt? O mî totuși simplă constă în măsurarea poziției din șir la care s-a făcut ultima permutare a valorilor. Într-adevăr, dacă la o parcurgere a șirului s-a făcut la al k -lea element, atunci acesta și următoarele $n - k$ elemente sînt ordonate (dovedit acest lucru!). La următoarea parcurgere este deci să fie cel să căutăm perechile rău ordonate doar pînă la al $(k - 1)$ -lea element. Restrictiv algoritmul de ordonare descrescătoare care să includă și această optimizare.

c) Prezentăm în cele ce urmează o altă metodă de ordonare. Ea se bazează pe parcurgerea șirului de numere pentru fiecare poziție și plasarea la sfîrșitul parcurgerii în poziția respectivă a valorii definitive. În acest fel, la a j -a parcurgere, primele $j - 1$ poziții sînt bine ordonate, deci valoarea din celula a j -a se compară doar cu valorile din celulele de la $j + 1$ la n . De fiecare dată cînd se găsește că a j -a valoare este mai mică

deci cea cu care se compară, cele două valori se permută.
Deci pentru un sir de 4 numere vom avea următoarele momente mai importante în ordonare:



Să se scrie algoritmul în LPS bazat pe această metodă.

- d) Fie algoritmul de ordonare descrescătoare descris în figura 1.32 printr-o schemă logică. Metoda folosită se numește metoda „inserării directe”. Să se analizeze metoda și să se scrie algoritmul în LPS bazat pe această metodă.

13. Să se scrie algoritmul în LPS pentru adunarea a două matrice de valori întregi. În afara valorilor (date în ordinea liniilor), pe banda d' intrare se dau două numere întregi pozitive reprezentând numărul liniilor, respectiv al coloanei matricii.

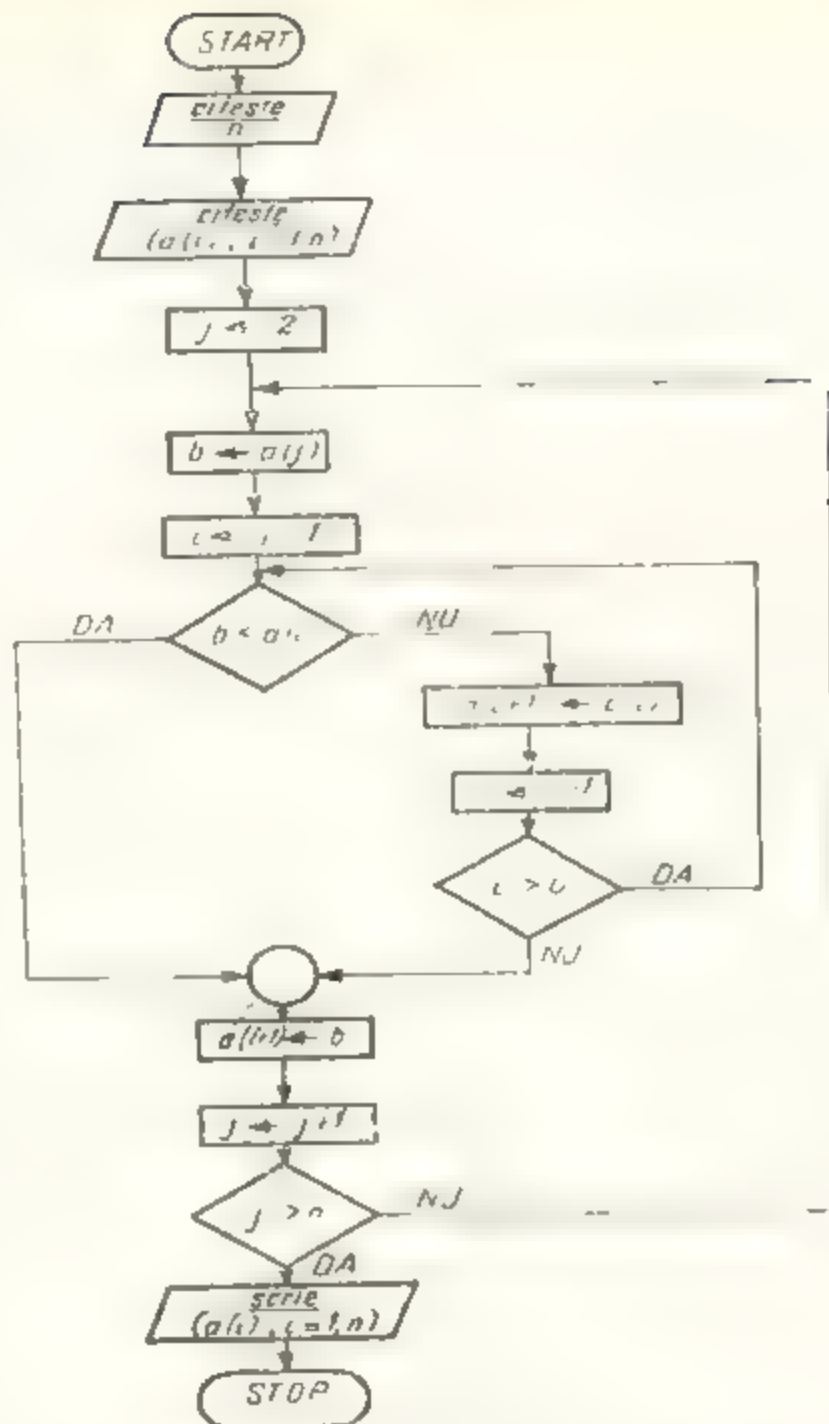


Fig. 1.32

Instrucțiunile de citire și scriere ale tablourilor cu două dimensiuni sînt extinderi ale notației folosite pentru tablourile cu o dimensiune. Să presupunem citirea tabloului a declarat ca în figura 1.33. Vom nota cu :

— $a(i, j)$ termenul general al tabloului a ,
 $(a(i, j), i = 1, n)$ o listă de elemente ale tabloului a aflate pe coloana j

adică : $a(1, j), a(2, j), \dots, a(n, j)$.

$((a(i, j), i = 1, n), j = 1, m)$ ne spune să repetăm de m ori lista $(a(i, j), i = 1, n)$ înlocuind pe j cu $1, 2, \dots, m$.

ϵ	var simplă	valori întregi [1,20]
π	var simplă	valori întregi [1,15]
α	tablou 2 dimensi (20,15)	valori reale

Fig. 1.33

Adică :

$$\underbrace{a(1, 1), a(2, 1), \dots, a(n, 1)}_{\text{prima coloană}},$$

$$\underbrace{a(1, 2), a(2, 2), \dots, a(n, 2)}_{\text{a doua coloană}}, \dots, \underbrace{a(1, m), a(2, m), \dots, a(n, m)}_{\text{a m-a coloană}}.$$

Să observăm că la citire ordinea celulelor în care se citesc valorile trebuie să reproducă ordinea numerelor de pe BI pentru ca încărcarea valorilor în celulele matrice să reproducă intenția utilizatorului. Aceleași notații le vom folosi și la scriere.

16. Un sistem de ecuații liniare de formă :

$$\begin{cases} a_{11}x_1 & = b_1, \\ \vdots & \\ a_{21}x_1 + a_{22}x_2 & = b_2, \\ \vdots & \\ a_{i1}x_1 + a_{i2}x_2 + \dots + a_{in}x_n & = b_i, \\ \vdots & \\ a_{n1}x_1 + a_{n2}x_2 + \dots + a_{nn}x_n & = b_n \end{cases}$$

se numește sistem triunghiular. Soluțiile lui se calculează pe rând astfel :

$$\begin{cases} x_1 = b_1/a_{11} \\ x_i = (b_i - \sum_{j=1}^{i-1} a_{ij}x_j)/a_{ii}, \text{ pentru } i = 2, \dots, n. \end{cases}$$

Să se scrie algoritmul în LPS care, primind la intrare numărul ecuațiilor și al variabilelor (cel mult 20), matricea coeficienților și vectorul termenilor liberi, tipărește la ieșire soluțiile sistemului.¹

17. Se dă pe banda de intrare o matrice pătrată de valori reale. Să se scrie algoritmul în LPS care înscrie pe banda de ieșire suma elementelor aflate în triunghiul superior al matricei mărginit de cele două diagonale.¹
18. Se dă un polinom prin gradul (valoare întreagă pozitivă) și coeficienții săi (valori reale).
- Să se scrie algoritmul în LPS care calculează valoarea polinomului într-un punct dat (valoarea reală).
 - Să se scrie algoritmul în LPS care calculează valoarea derivatei polinomului într-un punct dat (valoarea reală).

¹ Vezi indicația de reprezentare a tablourilor cu două dimensiuni de la problema 15.

- 19*. Se dă pe bandă de intrare un sir de valori reale precedat de numărul acestora (mai mic decât 1000). Să se scrie un algoritm în LPS care să tipărească pe bandă de ieșire toate valorile distincte din sir urmate de numărul lor de apariții în sir.

Indicație Se va folosi un tablou cu două coloane¹. În prima coloană se memorează numerele distincte, în a doua coloană numărul lor de apariții. Pentru fiecare număr citit de pe banda de intrare, el este căutat printre numerele primei coloane. Dacă este găsit crește cu 1 numărul de apariții; dacă nu este găsit se înscrie într-o nouă linie din tablou numărul citit și 1 ca număr de apariții. La sfârșitul citirii se tipărește tabloul.

20. Să se scrie algoritmul în LPS care să determine dacă un număr întreg pozitiv dat este număr prim sau nu, ieșirea va fi un mesaj: „da” sau „nu”.

Indicație Se împarte numărul, fie el n , la 2 și la toate numerele naturale de la 3 la $[n, 2]$ sau chiar $[n]$. Dacă măcar un rest este egal cu 0, numărul nu este prim.

21. Să se scrie algoritmul în LPS de generare a tuturor numerelor prime mai mici decât un număr întreg pozitiv dat.

22. Se scrie algoritmul în LPS care tipărește toți divizorii primi ai unui număr întreg pozitiv dat.

23. *Problema maimuței*. Pe o insulă pustie se află trei marinari lafragoși, și o maimuță. Maimuța a reușit să strângă o grămadă de nucă de cocos și să le împartă în trei părți egale, dar a dormit. Peste noapte, unul din marinari s-a sculat, a împărțit grămadă în trei părți egale după care rezultatul o nucă în plus a dat o maimuței, iar el a ascuns una din părți. După o vreme și povestea s-a repetat și cu ceilalți doi marinari. Dimineața, marinarii împart și grămadă rămasă în trei părți și din nou rămâne o nucă de cocos pe care o dau maimuței. Câte nucă de cocos au fost la început în grămadă? Deoarece rezultatul nu este unic, să se găsească toate valorile mai mici de 1000 care satisfac condiția.

Observații 1° Problema reprezintă un caz tipic pentru rezolvare cu ajutorul calculatoarelor, necesitând calcule relativ simple, dar laborioase.

2° Prelucrarea nu are date de intrare.

¹ Vezi indicația de reprezentare a tablourilor cu două dimensiuni de la problema 15.

Capitolul II

PROGRAMAREA ÎN LIMBAJUL FORTRAN

II.1. DATE FORTRAN. INSTRUCȚIUNEA DE ATRIBUIRE.

EXPRESII

II.1.1. Introducere

Asigurarea unei informări corecte, rapide și utilizabile în conducerea unei activități umane constituie, în ultimul timp, un deziderat de cea mai mare importanță. Ea se face simțită la toate nivelele societății, constituind unul dintre elementele determinante în obținerea unor rezultate bune în activitatea desfășurată. Conducerea activității unei uzine, stabilirea, planificarea, lansarea și urmărirea operațiilor de realizare a unui obiectiv (bloc de locuințe, fabrică, instalație industrială etc.), supravegherea unor procese complexe (lansarea și urmărirea evoluției unei rachete, comanda mașinilor-unelte sau a laminoarelor) constituie doar câteva din domeniile unde obținerea informațiilor prin prelucrarea datelor este de cea mai mare importanță. Unul din mijloacele cele mai perfecționate care îi oferă omului posibilitatea prelucrării unui volum mare de date, în timp util, în scopul obținerii informațiilor necesare este calculatorul electronic.

Existența fizică a calculatorului electronic, fie el oricât de perfecționat, nu răspunde necesităților sus-amintite. Este necesar să dispunem, de asemenea, de mijloace de comunicare adecvate între om și calculator - limbajele precum și de oameni care să poată folosi noile posibilități de prelucrare a datelor pentru soluționarea problemelor concrete - programatorii.

Limbajul FORTRAN constituie unul dintre mijloacele de descriere a prelucrărilor ce trebuie realizate de calculatorul electronic. Numele său provine din cuvintele englezești FORMula TRANslation care au semnificația de „traducerea formulilor”. Creat în 1956 pentru calculatoarele din prima generație acesta este și astăzi cel mai folosit limbaj. Aria sa de aplicabilitate s-a extins și la probleme care necesită în primul rând alte prelucrări decât calculele, așa numitele probleme economice. Acest limbaj poate fi învățat destul

de rapid, avînd o sintaxă și o semantică simple. Programele scrise în FORTRAN sînt traduse cu ajutorul unor compilatoare adecvate, codul rezultat fiind apropiat de codul mașina și el poate fi executat de către calculatorul real. Alte avantaje ale limbajului FORTRAN sînt alcăturarea ușoară a programelor, depanarea rapidă, precum și asemănarea dintre descrierea calculului în FORTRAN și în practica curentă.

Pentru a asigura compatibilitatea programelor, limbajul FORTRAN a fost standardizat. Aceasta înseamnă ca sintaxa și semantica limbajului au fost fixate și ca orice compilator construit pentru un calculator trebuie să le respecte. Astfel, dacă pe calculatoare diferite există compilatoare care respectă standardul FORTRAN, programele pot fi rulate pe oricare dintre acestea, obținîndu-se aceleași rezultate. De asemenea, a fost posibilă construirea unor colecții de programe care oferă soluțiile unor probleme matematice des întîlnite în aplicații (rezolvări de ecuații sau de sisteme algebrice, calcule cu matrice, calcule statistice etc.). Aceste colecții — denumite *biblioteci de programe* — pot fi folosite ușor de către toți programatorii interesați, reducîndu-se astfel timpul și efortul necesare rezolvării problemelor.

Un alt avantaj al limbajului FORTRAN (ca, de altfel, al tuturor limbajelor de nivel înalt) constă în faptul că în acesta se programează mai rapid decît în limbajul de asamblare al calculatorului. Mai mult, cel ce programează nu are nevoie să cunoască limbajul de asamblare, iar despre structura calculatorului trebuie să aibă doar cunoștințe generale.

II.1.2. Etapele rezolvării unei probleme utilizînd limbajul FORTRAN

Descrierea prelucrărilor care vor fi efectuate în limbajul FORTRAN — *scrierea programului* — constituie prima etapă în rezolvarea unei probleme. Deși deosebit de LPS, vom regăsi și în FORTRAN elementele de bază ale acestuia. Totuși aceste elemente sînt specifice limbajului FORTRAN, lucru normal dacă ținem seama ca programele scrise vor fi rulate pe un calculator real, și nu pe un calculator fictiv.

Scopul alcătuirii programului FORTRAN — obținerea soluției unei probleme concrete — va fi atins în momentul *executării programului* folosind datele necesare. Aceasta poate fi considerată ultima etapă în rezolvarea unei probleme folosind calculatorul (vezi figura II.1). Totuși între scrierea programului și obținerea rezultatelor există o etapă destul de laborioasă în care se face *punerea la punct a programului*. În această etapă programatorul va căuta să verifice dacă programul său îndeplinește sau nu prelucrarea dorită. Astfel, dacă programul său va fi destinat rezolvării unor ecuații de gradul al III-lea, în această etapă se va verifica dacă el rezolvă corect anumite ecuații ale căror soluții sînt cunoscute.

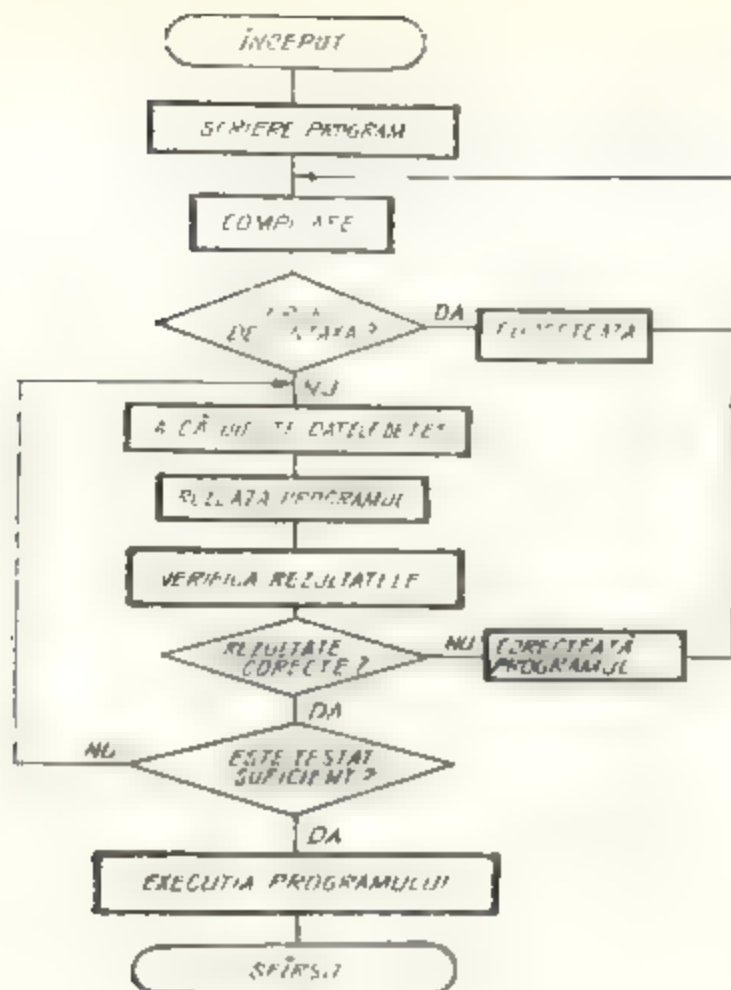


Fig. II.1

În etapa de punere la punct a programului, programatorul desfășoară o serie de activități, cu și fără ajutorul calculatorului. Dintre activitățile pentru care este nevoie de calculator, amintim compilarea programului și rularea programului cu date de test. Între activitățile descrise mai sus, programatorul corectează programul (înlătură erorile de sintaxă, schimbă algoritmul dacă este cazul etc.), depunează programul (identifică cauza erorii și o înlătură), alcătuiește datele de test, verifică rezultatele obținute ca urmare a rulării de test. Această etapă, punerea la punct a programului, este cea mai lungă în cadrul ciclului de elaborare a programului. De regulă se consumă pentru ea între 30% și 90% din timpul total al realizării programului care rezolvă problema respectivă

Scrierea programului. Instrucțiuni. Formularul FORTRAN

FORTTRAN fiind un limbaj de programare are reguli de scriere a programelor bine precizate și orice abatere de la ele conduce la semnalarea unor erori în momentul rulării pe calculator.

● Un program FORTRAN este alcătuit dintr-un șir de instrucțiuni. Instrucțiunile limbajului sînt formate din cuvinte care, la rîndul lor, sînt compuse din caractere. În FORTRAN pot fi folosite doar următoarele caractere¹:

- literele mari ale alfabetului, ABCDEFGHIJKLMNOPQRSTUVWXYZ
- cifrele : 0123456789
- semnele speciale : + - = / () ' * , și spațiu

● Instrucțiunile limbajului sînt de două feluri

- *executabile*, care cer îndeplinirea unei acțiuni efective, din care va fi generat programul în limbaj mașină (cînd nu se poate naște o confuzie le vom numi pe scurt instrucțiuni);

¹ Celelalte caractere care pot fi afișate de către un calculator particular (de exemplu, &, ≠ etc.) pot fi utilizate numai în date de tip șir de caractere. Spațiul îl vom nota în continuare cu □.

- *non-executabile* (de obicei acestea vor fi numite *declarații*), care descriu caracteristicile datelor utilizate, clasificarea programelor, indicații asupra formei datelor din seturile de date folosite etc.

● Structura unui program FORTRAN este de regula următoarea: în prima parte declarațiile, apoi instrucțiunile care descriu prelucrările efective care vor fi realizate asupra datelor descrise în partea de declarații iar ultima instrucțiune din program trebuie să fie întotdeauna END.

Reamintim că *programul FORTRAN este descrierea unor prelucrări*. În instrucțiunile executabile descriu, în general, un pas sau o anumită parte a algoritmului care stă la baza prelucrării. Această parte a programului descrie prelucrarea propriu zisă. Modul în care, pe baza instrucțiunilor, se îndeplinește prelucrarea dorită este următorul:

- se stabilesc atributele datelor utilizate, conform declarațiilor;
- se realizează acțiunea cerută de prima instrucțiune executabilă; dacă instrucțiunea curentă (în curs de execuție) nu indică executarea obligatorie a unei alte instrucțiuni se va îndeplini în continuare acțiunea cerută de instrucțiunea care urmează în program.

● În FORTRAN *forma generală a unei instrucțiuni* este următoarea:

<etichetă> || <instrucțiune>

<etichetă> este un număr întreg cuprins între 1 și 99999 și are drept scop identificarea unei instrucțiuni în cadrul programului. Acest număr îl vom numi etichetă. Trecerea la execuția altei instrucțiuni decât cea care o urmează pe aceea în curs de execuție se realizează cerînd în program să se execute obligatoriu instrucțiunea cu eticheta respectivă.

Eticheta marchează deci o anumită instrucțiune a programului, atașîndu-i un număr, pe baza căruiua poate fi identificată. Trebuie precizat că nu este necesar ca orice instrucțiune să aibă etichetă. În același timp, două instrucțiuni nu pot avea aceeași etichetă, lucru evident datorită scopului folosirii acestora – identificarea instrucțiunii. În figura II 2 este dat un exemplu de program FORTRAN, care rezolvă o ecuație de gradul I al cărei coeficienți sînt citați de pe o cartelă.

● Instrucțiunile, care în FORTRAN au un format prestabilit, se pot scrie pe liniile unui formular de programare standard, împărțit în 80 de coloane (vezi fig. II 2). Pe fiecare linie a formularului se poate scrie o instrucțiune. Scrierea este linară, neadmițîndu-se notațiile sub linie sau deasupra ei ca A_1 sau B^4 , iar caracterele ce compun instrucțiunile trebuie scrise câte unul în fiecare coloană.

Pachetul de cartele rezultat din perforarea instrucțiunilor ce pe formular va alcătui *programul sursă* ce va fi apoi citit și compilat de calculator.

O linie a formularului este compusă din patru zone astfel:

1. Primele 5 coloane alcătuiesc *zona pentru etichete*.

FORTAN Formulă de programare

Program: Rezolvarea ecuației de gradul I		INDICATORI PERFORMANȚE										Fig. 1			
		Semne		()	=	+								
Programator		Pentru		Data: 17 II 80		Scriitor									

Linia	10	20	30	40	50	60	70	80	Observații
1	PROGRAMUL FORTAN								
2	PROBLEMA: ECUAȚIA DE GRADUL I								
3	REAL A, B								
4	READ (5, A, B)								
5	IF (A.EB.0) GOTO 1								
6	I = -B/A								
7	WRITE (108, I, X)								
8	GOTO 3								
9	IF (B.EC.0) GOTO 2								
10	WRITE (108, I, X)								
11	GOTO 3								
12	WRITE (108, I, X)								
13	STOP								
14	FORMAT (2F6.2)								
15	FORMAT (1F5.2, 1F6.2)								
16	FORMAT (1F5.2, 1F6.2)								
17	FORMAT (1F5.2, 1F6.2)								
18	FORMAT (1F5.2, 1F6.2)								
19	END								

Fig. II 2

Fig. II.3

Etichetă					6	7	8	9	10	11	12	13	14	15	16	17	18
1	2	3	4	5													
1	2	3	4	5													
1	2	3	4	5													
1	2	3	4	5													
1	2	3	4	5													
1	2	3	4	5													
1	2	3	4	5													
1	2	3	4	5													

Blancurile ce apar între cifre sînt ignorate după cum se vede în figura II.3. Cifrele unei etichete nu pot fi scrise în alaltă acestor zone.

2. Coloana a 6-a - de *continuare*. Dacă o instrucțiune nu încape pe o singură linie de formular, scrierea ei se poate continua pe linia următoare punînd un caracter diferit de zero sau blank, în coloana a 6-a. Sînt admise cel mult 19 linii de continuare pentru o instrucțiune. Zona de etichetă corespunzătoare liniilor de continuare trebuie să fie liberă. Un exemplu este dat în figura II.4.

Etichetă					coloană de continuare																														
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	71	72	73	74	75	76	77	78	79	80	
		1	2	3		X	1					A	L	F	A	+	L	B	L	I	C	*			R	4	5		1	2	1	D			
						6	A	M	A	/	1	2	5	*		3	+									R	4	5		1	2	2	7		
						(R	E	L	+	Y)															R	4	5		1	2	3	11	

Fig. II.4

3. În zona de *instrucțiuni* (între coloanele a 7-a și a 72-a) se descriu instrucțiunile. Blancurile care apar între cuvinte sînt ignorate de compilator, ele fiind folosite numai pentru a ușura citirea instrucțiunilor.

Exemplu. Instrucțiunile următoare sînt echivalente:

|| READ (105, 25) A, B, C

|| READ |||(105, 25)||||A, |||B||, |||C

4. Zona de *identificare* (coloanele 73-80) servește pentru numerotarea rîndurilor sau pentru scurte comentarii. Datorită faptului că e posibilă o amestecare a caracterelor programului sursă, este utilă o numerotare a lor. Informațiile conținute în această zonă nu sînt luate în considerare de către compilator, ele fiind doar listate pe imprimantă.

Comentarii. Pentru a face programul cit mai explicit se pot adăuga informații suplimentare privind notații sau etape ale procesului de calcul, cu ajutorul unor linii de comentarii. O linie de comentariu se deosebește de celelalte, prin faptul că are un „C” în coloana 1. Comentariile pot fi scrise apoi, din coloana 2 pînă în 80.

Linile de comentariu pot apărea oriunde în program. Ele nu influențează prelucrarea, fiind doar listate la imprimantă împreună cu programul.

Rularea programelor scrise în limbajul FORTRAN

Dupa ce programul sursă a fost perforat pe cartele este nevoie ca el să fi compilat, corectat, testat etc. Aceste activități se desfășoară, după cum am mai spus, cu ajutorul calculatorului.

Pentru a rula un program pe un calculator din seria FELIX C este nevoie ca programatorul să alcătuiască o lucrare în care, în afara programului sursă, se mai adaugă și cartelele de comandă care dau informații sistemului de operare asupra modului cum se dorește să fie executată lucrarea. Vom da în continuare cartelele de comandă pentru rularea unui program FORTRAN care conține datele de pe cititorul de cartele și scrie rezultatul pe imprimantă.

col. 1	col. 11
*	JOB <i>nume1</i> , AN <i>cont</i> PN <i>nume2</i>
*	COMPILE FORTRAN
<div style="border: 1px solid black; padding: 5px; text-align: center;">program sursă</div>	
*	LINK
*	RUN
<div style="border: 1px solid black; padding: 5px; text-align: center;">date</div>	
*	EOF
*	EOJ

Data *cont* va fi dată de centrul de calcul pe al cărui calculator se va face rularea. Datele *nume1* și *nume2* sînt fixate de programator reprezentînd numele lucrării și respectiv al programatorului.

11.1.3. Date FORTRAN

Programul descrie de fapt modalitatea de transformare a datelor de intrare în date de ieșire. Calculatorul va îndeplini, în momentul execuției, acțiunile cerute de instrucțiunile programului. Fiind un limbaj destinat descrierii prelucrărilor care implică calcule numeroase, în FORTRAN sînt create facilitățile necesare pentru definirea și utilizarea unei mari varietăți de date.

În FORTRAN datele sînt grupate în două categorii:

constante ale căror proprietăți și valori sînt deduse din formele de scriere;

variabile ale căror proprietăți sînt descrise într-o declarație iar valorile lor sînt atribuite prin instrucțiuni executabile.

Trebuie precizat că, dacă o constantă are o singură valoare în tot timpul execuției programului, în schimb o variabilă poate să și schimbe valoarea de mai multe ori. De asemenea, trebuie reamintit că o variabilă nu poate avea două valori în același timp.

Atributele datelor FORTRAN sînt *tipul și structura*. Tipurile permise pentru date sînt *întreg, real, precizie dublă, complex, logic și caracter* (denumit și literal sau Hollerith). Structurile admise sînt *variabilele simple și tablourile*.

Pentru a înțelege diferența dintre tipurile de date, precum și necesitatea folosirii lor, trebuie să pornim de la reprezentările externe ale datelor folosite în mod obișnuit de către om. Fiecărui tip de dată în forma externă îi corespunde o reprezentare, într-o formă internă, proprie calculatorului.

În general utilizatorul nu cunoaște forma internă de reprezentare a datelor. Dar el trebuie să folosească formele externe de reprezentare a datelor conform restricțiilor impuse de limbaj.

Forma internă de reprezentare a numerelor este diferită de la un calculator la altul. Compilatorul trebuie să stabilească pentru fiecare dată FORTRAN forma internă de reprezentare a valorii ei. Pentru a face acest lucru trebuie ca adesea toată regula generală să fie îndeplinită la descrierea datelor.

Într-un program fiecare dată folosită trebuie să fie de tip permis în FORTRAN. O dată nu poate fi de două tipuri diferite în același program.

În FORTRAN pot fi folosite constante de oricare din tipurile admise. În cazul acestora, forma externă apare în program și pe baza ei compilatorul va deduce tipul și valoarea constantei. Pentru variabile, valoarea poate fi afișată sau citită sub formă externă. În program se utilizează numai numele variabilei. *Numele unei variabile poate fi orice sir de litere sau cifre care începe cu o literă*, compus din cel mult 8 caractere. Cîteva exemple pot fi: A25, A1FA, B, C2345, C2346, C234X34.

Tipul variabilelor poate fi întreg, real, precizie dublă, complex și logic. Nu există variabile de tip caracter.

*Tablourile desemnează grupuri de variabile de același tip. Fiecare variabilă din tablou poate fi referită prin numele tabloului urmat, între paranteze rotunde, de indici, care arată poziția ei în cadrul grupului. Numele de tablou și alcatușele după aceeași regulă ca și cele de variabile. Astfel, dacă tabloul A are 2 linii și 3 coloane, elementele lui vor fi A(1,1), A(1,2), A(1,3), A(2,1), A(2,2), A(2,3). Pentru a desemna variabilele care sînt elemente ale unui tablou de variabile simple, le vom numi *variabile indexate*. Trebuie menționat că și în cazul tabloulor nu există posibilitatea de a folosi variabile indexate de tip caracter.*

În continuare, dam pentru fiecare tip de date permis, principalele caracteristici.

Tipul întreg. *Forma externă* Număr întreg cu sau fără semn. În reprezentare nu poate apărea punctul zecimal. Semnal + este subînțeles dacă întregul nu are semn. *Exemple* 1,1 - 2175 0 4598345. *Valoarea* v asociată formei externe este aceea a numărului zecimal reprezentat. *Ordinul de mărime* $|v| \leq 2^{31} \dots$ sau $|v| \leq 2147483647$. *Formă internă* Reprezentarea exactă a valorii numărului, în virgulă fixă.

Tipul real. *Forma externă* Numărul real în reprezentarea căreia apare de obicei punctul zecimal sau un exponent. Aceasta reprezentare se face în două forme:

1. În limbajul FORTRAN elid se folosește punctul zecimal pentru reprezentarea unei valori în formă externă, aceea are rol identic cu virgula zecimală așa cum se cunoaște.

Forma I. Șir de cifre conținând un punct zecimal ce separa partea întreagă de partea fracționară. Șirul poate fi precedat de semnul + sau -. Partea fracționară și partea întreagă nu pot să lipsească în același timp. *Exemple:* +45.567 -123.896 .789 -6781. 0 0

Valoarea asociată este aceea a numărului zecimal ca aceeași scriere.

Forma II. Este similară cu scrierea numerelor zecimale sub forma: $-1 a_1 a_2 \dots a_p . a_{p+1} \dots a_n \times 10^q$. În FORTRAN în locul notății 10^q se va folosi notăția Eq unde q este exponentul. Deci forma generală de reprezentare este $\langle \text{fracție} \rangle E \langle \text{exponent} \rangle$ unde $\langle \text{fracție} \rangle$ poate fi reprezentată ca o dată reală în forma I sau ca una de tip întreg. *Exemple:* 31458E-2 -234E23 +.1985E+ 16 2731849E-6

Valoarea reprezentată va fi $\langle \text{fracție} \rangle \times 10^{\langle \text{exponent} \rangle}$. *Ordinul de mărime* este $10^{77} \times |v| \times 10^{72}$. Reprezentarea este aproximativă reținându-se cel mult 7 cifre semnificative. Exponentul poate avea cel mult două cifre. **Forma internă** pentru cele două moduri de reprezentare este reprezentarea aproximativă a valorii numărului în virgula mobilă format scurt (de regula pe 4 baiți).

Tipul precizie dublă. *Forma externă.* Numărul real în reprezentarea descrisă la tipul real, cu numatoarele deosebite. În locul literei E se folosește litera D pentru $\langle \text{exponent} \rangle$, iar pentru *valoare* se pot păstra până la 16 cifre semnificative. Restul caracteristicilor coincid cu cele ale datelor de tip real.

Tipul complex. *Forma externă.* O pereche de numere reale, care reprezintă partea reală și, respectiv, partea imaginară. *Exemple:* (21 0,1 0) (2 7, -82 +86) (.12345, -179E -2).

Trebuie precizat că aceasta este reprezentarea constantelor de tip complex. În cazul valorii variabilelor de tip complex, parantezele nu mai sunt folosite, forma externă constând din două date de tip real. *Ordinul de mărime, forma internă, valoarea părții reale și a părții imaginare* se deduc ușor din faptul că se utilizează reprezentări de tip real.

Tipul logic. *Forma externă.* Valorile logice ADEVĂR și FALS sînt reprezentate în instrucțiunile FORTRAN sub forma TRUE și respectiv FALSE. În imprimanta tipărește T pentru TRUE și F pentru FALSE.

Forma internă. O zonă de memorie (de regula un cuvînt), al cărui conținut permite identificarea celor două valori logice.

Tipul caracter (literal sau Hollerith). *Forma externă.* O dată literală este reprezentarea unei informații literale (șir de caractere) pentru procesul de calcul.

În instrucțiunile FORTRAN șirul de caractere este scos în evidență prin includerea lui între apostrofuri sau scriind înaintea acestuia nH unde n este numărul caracterelor din șir. La imprimare șirul de caractere este tipărit fără apostrofuri sau nH.

Se pot utiliza toate caracterele pentru care există o reprezentare în calculator. Dăm mai jos cîteva exemple de informații literale care pot fi reprezentate sub forma unei date literale:

Șirul de caractere

Reprezentarea la iesire

SCHIMBUL DOI
CLASA A XI

'SCHIMBUL DOI' sau
'CLASA A XI'

Reprezentare în instrucțiuni

12HSCHIMBUL DOI
10HCLASA A XI

Forma internă. Fiecărui șir de caractere îi corespunde o zonă de memorie, conținutul acesteia fiind stabilit în funcție de data literală folosită.

II.1.4. DECLARAȚII FORTRAN

Constantele sînt folosite în programul sursă în forma externă pe care am prezentat-o anterior. Prin modul de scriere, constantele își definesc valoarea și tipul. Celorlalte date FORTRAN, trebuie să li se asocieze, într-un mod bine stabilit, un tip și o valoare. Modul cum unei date i se asociază o valoare (reamintim că o variabilă poate avea mai multe valori în timpul execuției unui program) va fi tratat în paragrafele următoare. Stabilirea unui tip asociat variabilei se face conform convenției predefinite, printr-o declarație explicită sau printr-o declarație implicită.

a) **Convenția predefinită** permite să se asocieze un tip variabilei, după următoarea regulă :

În cazul cînd nu se specifică nimic despre tipul unei variabile, dacă prima literă a numelui este I, J, K, L, M sau N, tipul este întreg, dacă este orice altă literă, tipul este real.

Folosind convenția predefinită, programatorul poate alege numele variabilelor reale și întregi în așa fel încît să nu folosească declarații de tip în programul alcătuit.

Pentru a lucra cu variabile de celălalte tipuri sau a specifica un alt tip de variabilă decât cel rezultat din convenția predefinită se folosește declarația de tip.

Aceste declarații pot fi explicite sau implicite.

b) **Declarația explicită** precizează tipul asociat unor variabile din programul respectiv.

Forma generală a declarației explicite este :

$$|| t v_1, v_2, \dots, v_n$$

unde :

● t indică tipul și este unul din cuvintele INTEGER, REAL, DOUBLE PRECISION, COMPLEX sau LOGICAL.

● v_1, v_2, \dots, v_n este o listă de variabile care are cel puțin un element. *Semnificație* Variabilelor din listă li se asociază tipul t .

Lista reprezintă un șir de elemente separate prin virgulă. După ultimul element al listei nu trebuie pus nici un semn de punctuație.

Exemple :

Listă de numere :	41, - 2, 3, 0 - 2 20
Listă de variabile :	K1, K2, ALFA, MAX, Z
Listă de litere :	A, B, L, M, X, Y, Z

Trebuie reținut că declarațiile explicite nu pot avea etichete.

Exemple :

```
|| LOGICAL L1, L2, FALS
|| REAL MAX, INT, LUNG, R1, Z1, Z2
|| INTEGER ORE, ZILE, ANI, NUL
|| COMPLEX D, E
```

c) **Declarația implicită** este asemanătoare cu convenția predefinită, stabilirea tipului variabilei făcîndu se după prima literă a numelui.

Forma generală :

$$|| IMPLICIT t_1(l_1), t_2(l_2), \dots, t_n(l_n)$$

unde :

• t_1, t_2, \dots, t_n sînt cuvinte din mulțimea **INTEGER, REAL, DOUBLE PRECISION, LOGICAL și COMPLEX**.

• t_1, \dots, t_n liste ale căror elemente pot fi litere sau grupuri de două litere unite cu linuță (prima literă din grup trebuie să o precedă pe a doua în alfabet)

Semnificație O variabilă al cărei nume începe cu o literă ce face parte din lista t_k ($1 \leq k \leq n$) este de tipul t_k dacă numele ei n-a apărut într-o declarație explicită

— Dacă elementul unei liste are forma $a-b$ unde a și b sînt litere ale alfabetului, atunci toate literele din alfabet cuprinse între a și b fac parte din lista respectivă.

Exemplu. Într-un program apare declarația :

```
|| IMPLICIT LOGICAL (I - R), COMPLEX (X - D, N, V), LOGICAL, PRECISION  
   (F, G)
```

Variabilele al căror nume încep cu literele :

- A, B, C, D, X sau Y sînt complexe, — E, H, S, T, U, V, W, Z sînt reale,
- I, M, N, O, P, Q, R sînt logice, — F, G sînt în precizie dublă,
- J, K sînt întregi.

Pentru stabilirea tipului asociat unei variabile se aplică următoarea regulă :

— Dacă numele variabilei apare într-o declarație explicită atunci tipul ei este corespunzător acesteia,

În caz contrar, tipul se stabilește după prima literă a numelui, dacă litera apare într-o declarație implicită atunci variabila are tipul stabilit de aceasta, altfel se aplică convenția predefinită.

De asemenea trebuie reținut :

1. Declarațiile de tip dintr-un program nu trebuie să fie contradictorii.

Exemplu :

```
|| REAL MAX, I, L1, IDENT  
|| COMPLEX C1, GAMMA, L3  
|| LOGICAL L1, IDENT  
|| . . . . .
```

Conform acestor declarații variabile L1 și IDENT ar trebui să aibă aceeași valoare, lucru nepermis în FORTRAN

2. Declarațiile trebuie să apară la începutul programului, înainte de introducerea variabilelor. Dacă se folosește o declarație implicită ea trebuie să fie prima instrucțiune din program.

11.1.5. Instrucțiuni de atribuire

Limbaajul FORTRAN este foarte potrivit pentru rezolvarea problemelor tehnice și științifice. În definiția oricărei probleme un rol important îl joacă formulele, care exprimă calculele ce trebuie făcute. În limbaajul FORTRAN echivalentul unei formule este **instrucțiunea de atribuire**. Forma generală a acesteia este :

|| <v> = <exp>

Unde <v> este o variabilă (simplă sau indexată) iar <exp> este o expresie.

Semnificația instrucțiunii de atribuire este cea obișnuită, atribuirea variabilei <v> valoarea rezultată din evaluarea expresiei <exp>. În exemplele următoare sînt prezentate câteva instrucțiuni de atribuire

A = 5.3 variabila A primește valoarea 5.3

B2 = 3.14*R/A variabila B2 primește valoarea 3.14*R/A

TAB(8) A-B2 variabila TAB(8) primește valoarea expresiei A B2
 LOG = .TRUE. variabila logică LOG primește valoarea ADEVĂR.

Ca și tipul variabilelor, valoarea expresiilor poate fi de unul din tipurile: întreg, real, precizie dublă, complex și logic. Dacă tipurile lui $\langle v \rangle$ și al expresiei $\langle exp \rangle$ coincid, atribuirea este imediată. În cazul în care ele sînt diferite, este făcută mai întâi o conversiune a valorii expresiei la tipul variabilei $\langle v \rangle$ și abia după aceea este făcută atribuirea. Dacă aceasta conversiune nu se poate face (vezi figura II.5) este evident că instrucțiunea de atribuire este incorectă.

exp	Tip expresie exp				
	I	R	PD	C	L
I	A	1	1		
R	1	1	1		
PD	1	1	1		
C				1	
L					1

Fig. II.5

I = întreg; R = real; C = complex,
 PD = precizie dublă; L = logic.

II.1.6. Expresii FORTRAN

Expresia FORTRAN constă, reprezentată în ea est limbaj a unei expresii matematice obișnuite. Ea este o combinație a operatorilor, a operandilor și a parantezelor. Pentru a descrie modul de alcătuire a unei expresii FORTRAN corecte, vom analiza mai întâi care sînt operatorii și operandii permisi.

a) Operatorii FORTRAN care pot fi utilizați sînt de trei tipuri: aritmetici, relaționali și logici.

OPERATORI

ARITMETICI		LOGICI		RELAȚIONALI	
FORTRAN	Matematică	FORTRAN	Matematică	FORTRAN	Matematică
+	+	.OR.	sau	.LT.	<
-	-	.AND.	și	.GT.	>
*	x	.NOT.	negație	.EQ.	=
/	/			.LE.	≤
**	ridicare la putere			.GE.	≥
				.NE.	≠

Tabloul funcțiilor FORTRAN standard¹

Funcția	Definiție	Numele funcției în FORTRAN	Nr. argumentelor	Tip	
				Argumente	Valoare funcției
Valoare absolută	$ a $	ABS IABS DABS CABS	1 1 1 1	R I D C	R I D R
Modul	$a_1 = \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} a_2$	MOD AMOD DMOD	2 2 2	I R D	I R D
Obținerea părții reale a unui argument complex		REAL	1	C	R
Obținerea părții imaginare a unui argument complex		AIMAG	1	C	R
Expre mă două argumente reale în formă complexă	$a_1 + a_2\sqrt{-1}$	CMPLX	2	R	C
Obținerea conjugatului unui număr complex		CONJG	1	C	C
Exponențială	e^a	EXP DEXP CEXP	1 1 1	R D C	R D C
Logaritm natural	$\ln x$	ALOG DLØG CLOG	1 1 1	R D C	R D C
Logaritm zecimal	$\lg a$	ALOG 10 DLØG 10	1 1	R D	R D
Sinus	$\sin a$	SIN DSIN CSIN	1 1 1	R D C	R D C
Cosinus	$\cos a$	CØS DCØS CCØS	1 1 1	R D C	R D C
Arc tangent	$\arctg a$	ATAN DATAN	1 1	R D	R D
	$\arctg (b/a)$	ATAN 2 DATAN 2	2 2	R D	R D
Rădăcina pătrată	\sqrt{a}	SQRT DSQRT CSQRT	1 1 1	R D C	R D C

¹ Notăție: C - complex, I - întreg, R - real, D - precizie dublă

b) Operanzi FORTRAN pot fi :

- Constante sau variabile (simple sau indexate)
- Anumite funcții uzuale din matematică (radăcină pătrată, funcții trigonometrice, logaritmi etc.) Ele pot fi folosite în expresiile FORTRAN, în mod asemănător ca în matematică. Forma generală de descriere a utilizării unei asemenea funcții (numita apel sau chemarea funcției) este

nume funcție (argument₁, argument₂, ..., argument_n)

dacă presupunem că funcția are n argumente. Când o asemenea descriere apare într-o expresie FORTRAN înseamnă că valoarea funcției va fi calculată pentru argumentele indicate și va fi folosită drept operand la evaluarea expresiei. În figura II.6 sunt date funcțiile matematice care se pot folosi în expresiile FORTRAN. Modul de alcătuire a apelului funcțiilor, ca și tipul valorii rezultate din aceeași figură. Dăm mai jos câteva exemple :

ABS(ALFA)	$ \alpha $	ALOG10(B)	$\log_{10} b$
SQRT(6.25)	$\sqrt{6.25}$	ATAN (BETA)	$\arctg \beta$

- Expresii FORTRAN închise între paranteze. Acest tip de operand îl vom numi *subexpresie*.

Exemple: (B+C) (-B + SQRT(B*B + 4*A*C)) (2*A)

În concluzie, într-o expresie se pot folosi drept operanzi constante, variabile, funcții sau subexpresii.

Fiecărui operand i se asociază un tip conform figurii II.7. Acest tip este important în evaluarea expresiei, după cum vom vedea.

c) Evaluarea expresiilor

Expresiile pot fi împărțite în două clase, aritmetice și logice. Pentru expresiile aritmetice valoarea expresiei poate fi de tip întreg, real, precizie dublă sau complex. Tipul valorii expresiilor logice poate fi numai logic. Regula după care se găsește valoarea unei expresii le vom da pentru cele două clase, separat.

Operandul	Se asociază tipul
Constantă variabilă element tablou funcție subexpresie	Constantei variabilei tabloului funcției evaluării subexpresiei

Fig. II.7

$A \oplus B$		Tipul operandului B			
Tipul operandului A		I	R	PD	C
	I	I	R	PD	C
	R	R	R	PD	C
	PD	PD	PD	PD	C
	C	C	C	C	C

$A ** B$		Tipul operandului B			
Tipul operandului A		I	R	PD	C
	I	I	R	PD	C
	R	R	R	PD	/
	PD	PD	PD	PD	/
	C	C	/	/	/

LEGENDA \oplus una din operațiile $+$, $-$, $*$, $/$ I = întreg R = real
PD = precizie dublă C = complex

Fig. II.8

Expresii aritmetice

Rezultatul unei operații aritmetice depinde de tipul operandilor (aceștia nu pot fi de tip logic) așa cum rezulta din figura II.8 pentru adunare, scădere, înmulțire, împărțire și pentru ridicarea la putere.

Evaluarea expreselor aritmetice este făcută după reguli precis stabilite. Vom da în primul rând regulile după care sunt evaluate expresiile aritmetice fără paranteze (cele care nu conțin subexpresii).

S1 Dacă în expresie sunt folosite funcții, valorile acestora sunt găsite și se substituie apelurilor respective.

S2 Operațiile aritmetice sunt efectuate conform următoarelor priorități:

- 1) $**$
- 2) $*$ și $/$
- 3) $+$ și $-$

— În caz de egalitate a priorității operațiilor, efectuarea operațiilor aritmetice se face de la stînga la dreapta, cu excepția ridicării la putere care se execută de la dreapta la stînga:

$A**B**C$ se evaluează cu $A**(B**C)$

S3 Pentru efectuarea unei operații dacă operandii nu sînt de același tip se convertește valorile acestora în tipul rezultatului conform figurii II.8. Această regulă nu se aplică în cazul ridicării la o putere întregă a unui număr de alt tip: exponentul acestuia va rămîne de tip întreg.

Exemplul 1. Într-un program sînt folosite declarațiile:

```
IMPLICIT DOUBLE PRECISION (D, P, S, T), COMPLEX (U, Z)
REAL G(50)
```

În descrierea modului cum sînt evaluate expresiile care urmează au fost folosite mai multe variabile intermediare, ale caror nume au fost astfel alese încît tipul lor să poată fi dedus din declarațiile de mai sus.

$A/B + C \cdot C \cdot F$	$G(10) + B/C -$	$\underbrace{\text{SIN(ALFA)} \cdot C}_{R1} + 1.567$
$\underbrace{A/B}_{R2} + C \cdot R1$	$G(10) + B/C -$	$\underbrace{R1 \cdot C}_{R2} + 1.567$
$R2 + \underbrace{C \cdot R1}_{R3}$	$G(10) + \underbrace{B/C}_{R4} -$	$R2 + 1.567$
$\underbrace{R2 + R3}_{R4}$	$G(10) + R3 -$	$R2 + 1.567$
Valoare	$R4 -$	$R2 + 1.567$
	$\underbrace{\hspace{10em}}_{R5}$	$+ 1.567$
	Valoare	

Evaluarea unei expresii aritmetice oarecare se face după regulile următoare :

R1. Orice expresie aritmetică care nu conține subexpresii se evaluează după regulile S1, S2 și S3.

R2. Subexpresiile conținute într-o expresie trebuie evaluate înainte de a interveni ca operanzi într-o operație aritmetică. Se începe cu evaluarea celei mai din interior subexpresii. După calculul valorii unei expresii se pune în locul ei valoarea găsită, continuându-se procedeul cu celelalte subexpresii. Când se ajunge la expresia dată să nu mai conțină subexpresii, evaluarea se face după regula R1.

Exemplul 2. Într-un program se află următoarele declarații :

```
IMPLICIT DOUBLE PRECISION (D, T-Z)
REAL F(25), Y
DOUBLE PRECISION X(100)
```

În cele ce urmează vom arăta cum se evaluează anumite expresii folosite în acest program. Variabilele ce vor reține valori intermediare ale calculelor vor fi de tipul acestor valori.

$(A + B \cdot (I/J)) \cdot (((M/N - A/(B - \text{SQRT}(C))) - X(7))/F(5))$
$\underbrace{(A + B \cdot I)}_{R1} \cdot (((M/N - (A/(B - \text{SQRT}(C))) - X(7))/F(5))$
$R1 \cdot (((\underbrace{M/N}_{R2}) - A/(B - \text{SQRT}(C))) - X(7))/F(5))$
$R1 \cdot ((\underbrace{I2 - A/(B - \text{SQRT}(C))}_{R2}) - X(7))/F(5))$
$R1 \cdot ((\underbrace{I2 - A/(B - R2)}_{R3}) - X(7))/F(5))$
$R1 \cdot ((\underbrace{I2 - A/R3}_{R4}) - X(7))/F(5))$
$R1 \cdot (\underbrace{\hspace{10em}}_{R5} / F(5))$
Rezultă : $R1 \cdot R5$

Expresii logice

În FORTRAN o expresie logică poate fi :

- O constantă logică (.TRUE. sau .FALSE.)
- O variabilă logică sau un element al unui tablou de tip logic.

c) Două expresii aritmetice unite printr-un operator de relație.

d) Expresii logice unite prin operatori logici.

Se pot alcătui expresii logice de forma : $exp_1 R exp_2$ unde exp_1 , exp_2 sînt expresii aritmetice iar R e un operator de relație. Valoarea expresiei este găsită astfel : se evaluează expresiile aritmetice și apoi se compară valorile lor în funcție de operatorul de relație care apare între ele, rezultînd o valoare logică.

Exemplu Variabilele A, B, M și N au valorile 2.0, 1.5, 2 și respectiv 3

Dăm mai jos cîteva exemple de expresii logice și valoarea lor

A.NE.B .TRUE.

A.EQ.(B + A) .FALSE.

M.GT.(N*M) .TRUE.

A.EQ.M .TRUE.

Cînd expresiile aritmetice nu sînt de același tip, valorile vor fi convertite în tipul de rangul cel mai mare dat de următoarea ordine : întreg, real, precizie dublă, complex (în cazul exemplului ultim, în real).

Dacă expresiile logice sînt alcătuite folosind operatorii logici, evaluarea lor se realizează în funcție de paranteze și ținînd seama de următoarele priorități : expresii aritmetice, expresii de relație, NOT, AND, OR.

Exemple : Fie variabilele :

A, B, C, reale cu valorile 2.0, -1.5 și respectiv -10.0

I, J întregi cu valorile 4 și respectiv -2

L1, L2, L3 cu valorile .TRUE., .FALSE. și respectiv .TRUE.

În acest caz următoarele expresii logice au valorile indicate :

A.GT.C AND .NOT.L2 .TRUE.

L1 AND .NOT.L2 AND L3 .TRUE.

L1 AND I2 OR .NOT.L3 .FALSE.

A.LT.B&C AND I2 OR I.NE.J .TRUE.

II 1.7. Exerciții

1. Care sînt etapele de rezolvare a unei probleme în FORTRAN ?
2. Care sînt particularitățile etapei de punere la punct a programelor ?
3. Care dintre exemplele de mai jos constituie constante în sensul definițiilor date :

a) -121 b) 491.2 c) 514° d) 10*2

e) 41891 f) S418 g) ALFA h) 15.

i) -563 j) -12 + 02 k) +12345678910

4. Scrieți următoarele cantități numerice sub formă de constante reale

1.259,4 ; -2,959 ; 40001 ; 10^{25} ; 0,0000001 ; 10^{-2} .

5. Reprezentați aceste valori (exercițiul 4) sub formă de constante în precizie dublă

6. Care dintre următoarele constante reale sînt scrise greșit ? De ce ?

- a) 230,0001 b) $-2E + 12$ c) $+4.3E1 - 3$ d) $-4.1E2.1$

7. Care dintre perechile de constante reale au aceeași valoare ?

- a) 517.9 + 517.9 d) 23,1 + 231E-1
b) 256.1 0.2561E+3 c) 8.0 8
c) 0.11.6 0.000001

8. Care dintre următoarele constante în precizie dublă sînt corecte :

- a) 4214D.05 d) 0.D2
b) 64.2156 DP.5 e) 0.D.4
c) -5.34.1D51 f) .D+2

Explicați cauza incorectitudinii celorlalte.

9. Ce valoare au următoarele constante complexe ?

$$\begin{aligned} & (4.21, 1.0) \quad (5.1E+2, 1.5) \quad (5.71E-1, -6.7) \\ & (9.2E+2, -9.2E+02) \quad (1., 2.) \quad (4E+41, -5E-51) \end{aligned}$$

10. Scrieți sub formă de constante complexe valorile :

$$\begin{aligned} & 4 + 2i, \quad 5.2 - 6i, \quad 1 + i, \\ & 5.3 - 2.7102i, \quad 1 - i, \quad 1 \end{aligned}$$

11. Care dintre constantele complexe de mai jos nu sînt corecte :

- a) (4.21, 5.1) b) (2.E+1, .2) c) (1,1)
d) (0., 0.) e) (0.1E-86, 05E+90) f) (0.1, 0)

12. Numele unor variabile FORTRAN de mai jos sînt greșit scrise. Care sînt ele și de ce ?

I,1*	-42	VALDARE	111	total
TOLERANȚA	K-23	VALDARE		
25X	V-25	BUNA		
ALFA	V _{MIXT}	A1B25		

13. În programul în care apar numele variabilelor de mai jos nu se află nici o declarație cu privire la tip. Indicați care este tipul fiecăreia.

REZULTAT, REST, E, MAI, VAX, IJ,
SØRT, SACI, MARCA, X, Y, X1, XY,
EPSILON, CICO, PEPSI, BIG, TETA;

14. Transcrieți algoritmul din capitolul 12 folosind nume de variabile FORTRAN conforme cu tipul valorilor pe care le vor conține acestea.

15. Fără să folosiți declarația de tip, schimbați următoarele nume în așa fel încît să fie de tip întreg, păstrînd pe cît posibil sensul cuvintelor.

SUMA, CÎT, VAL, REST, GRAD, TEMP, X, Y.

16. Indicați de ce sînt greșite următoarele instrucțiuni FORTRAN

- a) INTREG M1, ALFA, PIT,
b) DOUBLE PRECISION, ON, B,
c) REAL K2, A*, IC.

17. Identificați erorile comise la scrierea următoarei secvențe de instrucțiuni

```
|| REAL ION, MAX
|| IMPLICIT INTEGER (A-M, Z)
|| IMPLICIT INTEGER (X-T)
```

18. De ce e greșită următoarea instrucțiune ?

```
|| IMPLICIT INTEGER(A-T), REAL(M)
```


19. De ce nu acceptă compilatorul aceste declarații?

```
|| REAL, M, N, V1
|| INTEGER, Z, V1, ALFA
|| COMPLEX N, C1
```

20. Încercați să explicați de ce același compilator nu poate să lucreze pe două calculatoare diferite, deși programele FORTRAN au aceeași formă

21. Într-un program s-a folosit următoarea declarație:

```
|| IMPLICIT INTEGER (A-Z)
```

Cum putem lucra totuși cu tablourile următoare:

A real cu 10 linii și 15 coloane
D1 și CM complexe cu 13 linii și 5 coloane
MINV logic cu 14 elemente?

22. Se consideră tabloul real Y cu 5 linii și 3 coloane.

- Scrieți elementele acestuia parcurgând coloanele una după alta
- Indicați ordinea elementelor când parcurgem prima linie apoi a doua etc. Coincide șirul găsit cu cel de la punctul a)?

23. Este posibil ca toate elementele de pe linia a 5-a din tabloul real cu 15 linii și 10 coloane să aibă valori complexe? De ce?

24. Identificați erorile din următoarele grupuri de instrucțiuni:

- DIMENSION A(5, 1), B(9)
INTEGER MAX, A(6)
- REAL A(4, 2, 3, 5, 6, 7, 8, 10)
- DIMENSION ALFA (M, N)
- DIMENSION, A, B (10, 7)

25. Scrieți în FORTRAN următoarele expresii (iterele grieești reprezintă valori reale).

a) $(x+1)^{n+1} + x^{n+1}$,

b) $x^2 - 2x \cos \alpha + 1$.

c) $\frac{a}{2-a} \cdot \frac{1+a}{2-b}$,

d) abc ,

e) $\frac{\frac{1}{b}}{\frac{m}{c} + ab} \cdot \frac{1}{a}$,

f) $\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$

g) $\frac{a}{2-a} \cdot (1-a) \cdot \frac{1}{1-a}$,

h) $\frac{a}{1 - \frac{1}{b}} + \frac{1 - \frac{1}{c}}{\frac{-1}{1-a}} + \frac{1}{\frac{x+y}{2}}$,

i) $\frac{-\cos^2 \alpha \sin^2 \alpha}{p+1}$,

j) $\sqrt{p(p-a)(p-b)(p-c)}$.

26. Într-un program avem următoarele declarații:

```
|| IMPLICIT DOUBLE PRECISION (R-T), COMPLEX (U-Z), LOGICAL (II, L)
|| DIMENSION F(40), L(7), TAU(15)
```

Arătați cum sunt evaluate expresiile de mai jos:

- $B1 + M1/M2_{**}K_{*}C - A_{*}F(3)$
- $A_{*}B/C/D_{**}M1(15)/F(16)_{*}A2$
- $4.25_{*}R2 + M1/M2_{*}CAR_{**}T1$
- $MAX1(4.2, 5.E + 1, 49.99)_{*}ALFA/R1_{**}M$
- $U + V_{*}W/(2.5, 4.)_{**}2$
- $A.LT.R.AND..NOT.A.EQ.B_{**}2 + 4_{*}A_{*}C$
- $C_{*}(1 - C)_{**}2 + (-2./3)/C + 7$

27. Care sînt greșelile comise în scrierea următoarelor expresii (declarațiile de tip sînt aceleași ca la exercițiul precedent).

$$\frac{4M + N}{A + C/3D + 7}$$

28. Ce valori vor avea expresiile :

a) $4.2/2.1 * 15 - 3$

b) $4.1 * 2 * (2/5)$

c) $4.1 * 2 * (2.5)$

d) $(3 * (A ** (1/3)) + B ** (1/2)) / ((A - B) ** (6/10))$, unde A și B au valorile 3.71, și, respectiv, -2.73E + 5

29. Scrieți în FORTRAN următoarele formule :

a) $t = \frac{1}{xy} + \frac{1}{yz} + \frac{1}{zx}$

b) $v = \frac{b + c}{a} + \frac{c + a}{b}$

c) $x = \frac{ab}{a + b} - \frac{a}{b + c}$

d) $y = \sqrt{a + \sqrt{ab}}$

e) $z = \left(\frac{\sqrt{ab} - 5.421}{\sqrt{a} + \sqrt{c}} \right)^{1.2}$

f) $z = \frac{a^{-1/2}}{69} \sqrt{\frac{\sin a}{2}}$

g) $x = \frac{y^2 + 2}{b} \sin y$

h) $y = x^a z^b$

i) $z = 5^{xy} + abc^d$

30. Fie declarația

|| IMPLICIT COMPLEX (U-Z), DOUBLE PRECISION (P-T), LOGICAL (H, L)

Ce valori vor primi variabilele întrebuate în instrucțiunile de atribuire de mai jos ?

|| A = 12.3 - 5
H = 16/2
C = 2 * (12/3)
P = 5 * (1.5)
W = (2.2, 5.15) + (4.1, 2.5)
U = (1.0, -1.) * (2., 1.)
L1 = .TRUE. .AND. .FALSE.

|| IA = 5/2
A11 = 5/2
A2 = 1/3
T = 20/6 + 15/2
U1 = 5/2
I1 = 2 * (5/2)
A1 = 1.000 * (999/1.000)
M = 5.21D0 + 4.25

11.2. PRINCIPALELE INSTRUCȚIUNI ALE LIMBAJULUI FORTRAN

11.2.1. Introducere

Instrucțiunile «pseudolimbajului» LPS, studiate în capitolul precedent, cu ajutorul cărora s-au scris programe LPS, se pot grupa, în funcție de prelucrarea îndeplinită, astfel¹: instrucțiuni de intrare/ieșire, condiționale, de ciclare, de atribuire și de oprire.

În limbajul FORTRAN se folosesc pentru descrierea prelucrărilor tipuri de instrucțiuni similare cu cele de mai sus, dar care au o semantică și o sintaxă diferită, proprie limbajului. Astfel, în LPS terminarea textului unei in-

¹ Pentru a înțelege mai bine cele ce urmează, este bine ca sintaxa și semantica acestor instrucțiuni să fie reamintită

instrucțiuni de ciclare era indicată prin ■, în timp ce în FORTRAN nu avem un asemenea semn, ceea ce a făcut necesară, printre alte lucruri, introducerea și folosirea etichetelor.

După cum am mai spus, etichetele pot fi utilizate în zona corespunzătoare a instrucțiunilor FORTRAN, atunci când programatorul consideră necesară identificarea instrucțiunii respective. De regulă, fiecare instrucțiune FORTRAN poate avea sau nu etichetă, lucru pe care îl presupunem implicit în descrierile instrucțiunilor date mai jos. Vom preciza explicit doar cazurile de excepție: când apariția unei etichete în zona corespunzătoare este obligatorie sau dimpotrivă interzisă. În continuare vom studia principalele tipuri de instrucțiuni FORTRAN.

11.2.2. Instrucțiunea condițională

În FORTRAN sînt folosite două instrucțiuni condiționale, prelucrările exprimate de acestea fiind mai puțin puternice decît ale instrucțiunilor corespunzătoare tip din LPS.

11.2.2.1. Instrucțiunea IF logic

Forma generală:

|| IF(<expresie logică>) <s>

unde <s> este orice instrucțiune executabilă cu excepția lui DO sau a unei alte instrucțiuni IF logic.

Semnificație În momentul execuției acestei instrucțiuni, se evaluează valoarea expresiei logice. Dacă valoarea acesteia este «FALSE» atunci instrucțiunea <s> nu se execută continuîndu-se execuția cu instrucțiunea următoare, în caz contrar se execută instrucțiunea <s>.

11.2.2.2. Instrucțiunea IF aritmetic

Forma generală:

|| IF (<expresie aritmetică>) <e₁>, <e₂>, <e₃>

unde <e₁>, <e₂>, <e₃> sînt etichete.

Semnificație Pentru execuția acestei instrucțiuni se evaluează expresia aritmetică. În funcție de valoarea expresiei (negativă, zero sau pozitivă), se execută instrucțiunea cu eticheta <e₁>, <e₂> sau, respectiv, <e₃>.

Observație Un rol important îl au în descrierea prelucrărilor, etichetele care permit să se indice locul unde se va continua prelucrarea în funcție de valoarea unor expresii, de anumite condiții etc.

11.2.3. Instrucțiuni de salt necondiționat

Dacă într-o instrucțiune IF aritmetic toate etichetele sînt egale, execuția programului se va continua cu aceeași instrucțiune. Astfel

IF(A - 2) 33, 33, 33

va avea drept efect continuarea execuției cu instrucțiunea avînd eticheta 33, indiferent de valoarea expresiei A - 2.

Deși o asemenea posibilitate de programare poate fi folosită, ea nu este recomandată deoarece ar conduce la o înțelegere greoaie a prelucrării exprimate de program. Pentru a da posibilitatea programatorului să transfere controlul în orice punct al programului, limbajul FORTRAN îi pune la dispoziție instrucțiunea GØ TØ.

Forma generală :

|| GØ TØ <e>

unde <e> este o etichetă.

Semnificație După execuția acestei instrucțiuni controlul va fi trecut (deci se va executa în continuare) instrucțiunii cu eticheta e.

Observație Pentru a fi executată instrucțiunea care urmează după un GØ TØ sau un IF trebuie să aibă întotdeauna etichetă.

În secvența de instrucțiuni :

```

      || GØ TØ 72
      || IF(K) 73, 74, 75
72 || I      I + 1

```

instrucțiunea IF nu poate fi executată niciodată, ea fiind sărită întotdeauna datorită instrucțiunii GØ TØ. Pe de altă parte, nici o instrucțiune a programului nu poate cere execuția ei, deoarece nu poate fi identificată (nu are etichetă).

11.2.4. Instrucțiunile : CONTINUE, STOP, PAUSE și END

Forma generală a instrucțiunii CONTINUE

|| CONTINUE

Semnificație Instrucțiunea nu afectează secvența normală de executare a instrucțiunilor. Când este întâlnită se trece la executarea instrucțiunii care o urmează în program.

Forma generală a instrucțiunii STØP

```

|| STØP
|| STØP <n>

```

unde <n> e o constantă întreagă, fără semn, cu cel mult 5 cifre.

Semnificație Instrucțiunea pune în evidență sfârșitul execuției programului. Utilizarea formei a II-a va produce afișarea pe consola calculatorului a numărului <n>. Când într-un program se folosesc mai multe instrucțiuni STØP, știm astfel la care s-a ajuns, când a luat sfârșit execuția programului.

Forma generală a instrucțiunii PAUSE

```

|| PAUSE
|| PAUSE <n>
|| PAUSE <mesaj>

```

unde <n> e o constantă întreagă, fără semn, cu cel mult 5 cifre.

Semnificație. Se folosește pentru a opri temporar execuția unui program. Reluarea programului se face la comanda operatorului. Această instrucțiune se folosește fie pentru a cere o acțiune manuală a operatorului (schimbarea unei benzi, a unui disc etc.) fie pentru scrierea unor informații despre programul ce se execută (delimitarea unei erori etc.). Dacă se folosește forma a doua sau a treia, pe lângă cuvântul PAUSE care apare scris pe consolă, se adaugă și numărul <n>, respectiv mesajul indicat, astfel încât se poate identifica care a fost instrucțiunea PAUSE ce a întrerupt execuția programului.

Forma generală a instrucțiunii END

|| END

Semnificație. Este o instrucțiune neexecutabilă. Este folosită pentru a indica compilatorului care e ultima instrucțiune din programul sursă. Ea trebuie să apară în fiecare program. Această instrucțiune nu poate avea etichetă.

11.2.5. Instrucțiuni de intrare/ieșire

În LPS valorile variabilelor în forma externă erau reprezentate pe banda de intrare și banda de ieșire. Folosind instrucțiunea **citește** v valorile erau citite de pe banda de intrare și transferate în memoria principală. În mod similar valoarea unei variabile era transferată din memorie pe banda de ieșire prin instrucțiunea **scrie x**.

Pentru un calculator fictiv acest lucru era suficient. În cazul unui calculator real, datele pot fi reprezentate pe diverse medii externe ca, de exemplu: cartelă perforată, hârtie de imprimantă, bandă magnetică sau disc magnetic. Aceasta complică foarte mult problema transferurilor de date spre și dinspre memoria principală.

Descrierea unui schimb de date între o unitate externă și memorie, deci a unei operații de intrare/ieșire, presupune precizarea

- operației ce trebuie efectuată (citire sau scriere) ;

- variabilelor ale caror valori sînt implicate în transferul respectiv (prin aceasta este precizată și forma internă de reprezentare) ;

- locul unde se găsesc datele reprezentate în forma externă (operația de intrare/ieșire se va îndeplini între această unitate și memorie) ;

- formei de reprezentare externă a datelor (structura înregistrării care va fi citită sau scrisă).

Pentru ca operația de intrare/ieșire să fie cât mai independentă de dispozitivul fizic particular care va fi implicat în transferul de date, limbajul FORTRAN presupune că datele sînt organizate pe mediul extern sub formă de înregistrări. Mulțimea înregistrărilor de pe un mediu extern alcătuiește un set de date pe care adesea îl vom numi fișier. Fiecare set de date va primi un număr care-l va identifica. El va fi indicat în fiecare operație de intrare/ieșire. În momentul execuției se va preciza ce dispozitiv fizic corespunde acestui număr. *Exemplu :* dacă unui set de date pe cartele (înregistrarea este cartela de 80 de coloane) îi asociem numărul 105 atunci în toate operațiile de citire ce le vom îndeplini pentru acest set de date vom indica acest număr.

Fizic, setul de date se va prezenta sub forma unui teanc de cartele pe care sînt perforate unele date în formă externă. Abia în timpul execuției se va arata pe ce cititor de cartele este plasat acest teanc.

Din punctul de vedere al limbajului FORTRAN, fiecare înregistrare se prezintă sub forma unui șir de caractere. Forma în care aceste caractere sînt grupate în cîmpuri (zone), care constituie reprezentările valorilor unor variabile, este precizată într-o descriere a structurii acestei înregistrări, dată prin instrucțiunea **FORMAT**. Această instrucțiune este atașată unei instrucțiuni de ieșire/întare.

Variabilele care participă la un transfer de date sînt precizate printr-o *listă de intrare/ieșire*. Deși au aceeași formă, interpretarea lor este diferită în funcție de operație: intrare sau ieșire.

O *listă de intrare* precizează variabilele simple sau indexate carora le sînt atribuite valori printr-o operație de intrare.

O *listă de ieșire* precizează variabilele ale căror valori sînt luate din memorie și scrise pe un mediu extern.

În listele de intrare/ieșire pot apărea nume de tablouri. În acest caz se presupune că toate elementele tabloului sînt implicate în operația respectivă, în ordinea așezării lor în memorie. În FORTRAN elementele unui tablou sînt așezate în locații succesive de memorie, mărindu-se la început valoarea primului indice pînă ajunge la dimensiunea maximă, apoi a celui de al doilea etc. De exemplu, elementele tabloului $Z(2, 3, 2)$ sînt plasate în memorie în ordinea din figura II.9. Dacă în lista de intrare/ieșire este indicat un element al unui tablou (variabilă indexată) atunci numai acel element participă la transfer.

$Z(1, 1, 1)$
$Z(2, 1, 1)$
$Z(1, 2, 1)$
$Z(2, 2, 1)$
$Z(1, 3, 1)$
$Z(2, 3, 1)$
$Z(1, 1, 2)$
$Z(2, 1, 2)$
$Z(1, 2, 2)$
$Z(2, 2, 2)$
$Z(1, 3, 2)$
$Z(2, 3, 2)$

Fig. II.9

II.2.5.1. Instrucțiunea de citire

Forma generală a instrucțiunii **READ**

READ ($\langle u \rangle$, $\langle f \rangle$, **END** $\langle e \rangle$) \langle listă intrare \rangle

unde $\langle u \rangle$ este numărul setului de date (pentru calculatorul FELIX-C numărul 105 desemnează unitatea standard de intrare cititorul de cartele);

$\langle f \rangle$ — eticheta instrucțiunii **FORMAT** asociată instrucțiunii **READ**;

$\langle e \rangle$ — eticheta unei instrucțiuni de la care se va continua prelucrarea în cazul în care s-a cerut citirea valorilor unor variabile, dar în setul de date nu mai sînt înregistrări.

Semnificație — Prin execuția acestei instrucțiuni, sînt atribuite valori variabilelor din lista \langle listă intrare \rangle , valori care sînt citite din una sau mai multe înregistrări ale setului de date cu numărul $\langle u \rangle$. Caracteristicile valorilor și aranjarea lor în înregistrările de pe mediu extern sînt descrise în instrucțiunile **FORMAT** cu eticheta $\langle f \rangle$. Folosirea opțiunii **END** $\langle e \rangle$ nu este obligatorie; dacă ea nu este folosită și nu se pot atribui valori tuturor variabilelor din listă-intrare, deoarece nu mai sînt înregistrări pe setul de date $\langle u \rangle$, este semnalată o eroare și prelucrarea se oprește; în situația cînd este folosită, se continuă prelucrarea cu instrucțiunea cu eticheta $\langle e \rangle$.

II.2.5.2. Instrucțiunea de scriere

Forma generală a instrucțiunii WRITE

WRITE (<u>, <f>) <listă-ieșire>

unde : <u> reprezintă numărul setului de date de ieșire (pentru calculatorul FELIX-C numărul unității standard de ieșire — imprimanta — este 108);
<f> eticheta instrucțiunii FORMAT asociate;
<listă-ieșire> — lista de ieșire.

Semnificație : Variabilele din lista de ieșire <listă ieșire> vor fi scrise pe setul de date cu numărul <u>, sub forma de înregistrări al căror format este specificat în instrucțiunea FORMAT cu eticheta <f>.

Exemplu : În acest moment avem cunoștințele necesare pentru a înțelege programul din figura II.2. Acest program, după cum arată comentariul din linia 10 (vezi zona de identificare), găsește soluția unei ecuații de gradul I. Algoritmul după care se rezolvă ecuația cu coeficienți A și B este cel din figura I.20, descris în LPS. Vom descrie pe scurt liniile programului FORTRAN. După declararea variabilelor reale A, B și X (linia 20) urmează citirea coeficienților A și B ai ecuației (linia 30) dintr-o cartelă folosind instrucțiunea FORMAT cu eticheta 11. În continuare (linia 40) prin instrucțiunea IF logic se testează dacă valoarea lui A este zero : dacă acest lucru este adevărat se sare necondiționat la instrucțiunea cu eticheta 1. Dacă A nu are valoare nulă se continuă cu calculul soluției X (linia 50), scrierea ei pe imprimantă cu FORMAT-ul 12 (linia 60) și saltul necondiționat prin instrucțiunea GO TO la instrucțiunea cu eticheta 3 care indică sfârșitul prelucrării (linia 120) deoarece ecuația a fost rezolvată. Dacă B = 0 se sare la instrucțiunea cu eticheta 2, unde se scrie pe imprimantă că avem o infinitate de soluții, după care se oprește prelucrarea. Ultima secvență, care se execută când B e diferit de zero, afișează pe imprimantă mesajul «nici o soluție». O ultimă observație este : ca între sfârșitul prelucrării (linia 120) și sfârșitul textului programului (linia 170) se află mai multe instrucțiuni neexecutabile.

Exerciții : Căutați să identificați, în descrierea de mai sus, liniile sursă care descriu prelucrările arătate, dar ale căror numere nu au fost precizate. Identificați elementele componente ale fiecărei instrucțiuni cunoscute.

II.2.6. Instrucțiunea FORMAT

Instrucțiunea FORMAT este o instrucțiune neexecutabilă, care dă informații despre structura înregistrărilor dintr-un set de date. În felul acesta compilatorul va deduce unde și cum sînt reprezentate valorile variabilelor din lista de intrare/ieșire în înregistrare. O înregistrare se compune din mai multe cîmpuri, fiecare cîmp putînd conține forma externă a unei valori numerice sau alte date (texte, spații etc.). Instrucțiunea FORMAT va trebui să ofere informații despre fiecare din cîmpurile înregistrării. Ea face acest lucru prin intermediul descriptorilor. De asemenea, prin intermediul unei instrucțiuni se pot descrie mai multe înregistrări.

Forma generală a instrucțiunii FORMAT este :

<etichetă> || FORMAT (<listă-descriptori>)

Semnificație - Descriptorii care apar în <listă descriptori> (numita în continuare și lista **FORMAT**) vor arăta care este structura unei înregistrări. Aceste instrucțiuni pot apărea oriunde în program (nu înainte de instrucțiunea **IMPLICIT** sau după instrucțiunea **END**) și trebuie să aibă etichetă. Între lista de intrare/ieșire și lista de descriptori se stabilește o corespondență astfel: fiecărei variabile din lista de intrare/ieșire i se asociază un descriptor de reprezentare a valorii acesteia.

În continuare, vom analiza următoarele categorii de descriptori: de reprezentare a valorilor variabilelor, de așezare a cîmpurilor în înregistrare și a înregistrărilor în cadrul setului de date.

II.2.6.1. Descriptori de reprezentare a valorilor variabilelor

Fiecărei variabile sau fiecărui element de tablou din lista de intrare/ieșire trebuie să le corespundă în lista **FORMAT** câte un cod care va descrie cîmpul din înregistrare de unde se citesc sau unde se scriu valorile asociate.

Vom descrie în continuare descriptorii utilizați pentru manipularea valorilor întregi și reale. Pentru a insera explicațiile la descrierea fiecărui descriptor, presupunem ca variabila careia acesta i-a fost asociat este *var*, iar cîmpul în care valoarea este reprezentată ocupă *w* poziții.

● **Descriptorul I pentru transmiterea valorilor variabilelor de tip întreg**
Forma generală: **Iw**

Semnificație - litera **I** arată că variabila *var* careia i s-a asociat acest descriptor trebuie să fie de tip întreg.

La intrare valoarea din cîmpul de *w* poziții al înregistrării de intrare este citită, se efectuează conversiunea la forma internă și rezultatul este atribuit variabilei *var*. Șirul de caractere din cîmpul de intrare trebuie să aibă forma unei constante întregi cu sau fără semn. Dacă semnul nu este pas, valoarea se consideră pozitivă. Dacă în acest cîmp apar spații ele sînt considerate zerouri.

La ieșire valoarea variabilei întregi corespunzătoare din lista de ieșire va fi afișată sub forma unui număr întreg, zecimal, aliniat la dreapta în cîmpul de *w* poziții.

Afișarea se face cu semnul - dacă valoarea e negativă și fără semn dacă e pozitivă. Dacă cele *w* poziții nu sînt suficiente pentru a afișa valoarea variabilei *var* în formă externă, în cîmpul destinat afișării vor fi scrise asteriscuri.

Exemplul 1 Dacă variabila **K** are valoarea -1705 și folosim pentru afișare formatul **I7** va avea reprezentarea din figura II.10

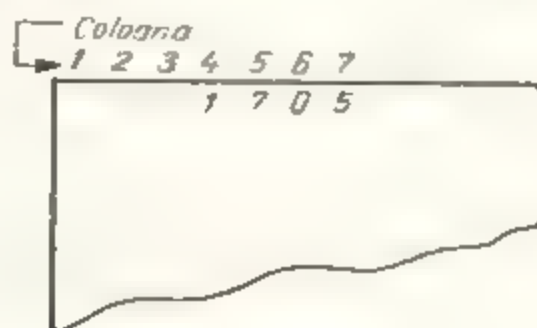


Fig. II.10

Exemplul 2 În figura II.11 sînt reprezentate diferite imagini ale unor cartele de pe care se pot citi, cu codul indicat în dreptul acestora, valori egale pentru variabilă: -70209

Cod folosit	Imaginea primelor coloane ale cartelei
16	7 0 2 0 9
17	7 0 2 0 9
17	7 0 2 0 9
17	7 2 0 9
18	7 0 2 0 9
18	7 2 9
19	7 0 2 9

Fig. II.11

● **Descriptorii pentru transmiterea valorilor variabilelor reale și precizie dublă**

Cîmpul unde este reprezentată valoarea unei astfel de variabile se poate descrie cu oricare din codurile E, F sau D.

Forma generală a acestor coduri este :

$$Fw.d \quad Ew.d \quad Dw.d$$

unde w indică cîte caractere are întregul cîmp în care este reprezentată valoarea, iar d numărul cifrelor zecimale din reprezentare. Este evident că trebuie să avem $w \geq d$.

Semnificația de intrare : Reprezentarea valorii variabilei în cele w poziții se poate face folosind sau nu punctul zecimal. Folosind punctul zecimal se reprezintă valoarea sub forma unei constante reale. Toate blancurile ce apar în reprezentare sînt considerate drept zero-uri.

— Dacă nu folosim punctul zecimal în reprezentarea valorilor ce vor fi citite, o importanță hotărîtoare o va avea indicația dată de d . Astfel, cînd în cîmpul de w poziții se afla un șir de cifre zecimale cu sau fără semn și fără exponent zecimal, valoarea transmisă va fi găsită considerînd că ultimele d poziții din cîmp constituie partea zecimală.

Dacă apare un exponent zecimal, iar șirul de cifre care îl precede nu conține punctul zecimal, atunci primele d poziții din fata exponentului sînt considerate ca alcătuind partea zecimală.

Semnificația la ieșire : Afișarea valorii oricărei variabile reale sau în precizie dublă se poate face cu oricare din cele trei coduri, E, F, D. După ce valoarea a fost convertită în zecimal, reprezentarea va avea o formă ce depinde numai de codul folosit.

Descriptorul E. Valoarea variabilei se va afișa sub una din formele :

$$X_1 X_2 X_3 \dots X_d E \pm Y_1 Y_2 \text{ dacă valoarea este pozitivă}$$

$$\text{-- } X_1 X_2 X_3 \dots X_d E \pm Y_1 Y_2 \text{ dacă valoarea este negativă}$$

unde :

$$X_1, X_2, \dots, X_d, Y_1, Y_2 \text{ sînt cifre zecimale și } X_1 \neq 0$$

Așadar, w trebuie să conțină 4 poziții pentru partea de exponent zecimal, d poziții pentru partea fracționară, o poziție pentru punctul zecimal și o poziție pentru semn. Deci, pentru ca reprezentarea externă a valorii unei variabile să fie posibilă în codul E, trebuie să fie adevărată relația :

$$w \geq d + 6.$$

Exemplu. Fie secvența :

```
25 | | ALFA = - 409 79  
    | | WRITE (4,25) ALFA  
    | | FORMAT (F11.5)
```

Prin aceasta este scrisă pe unitatea externă, identificată cu numărul 4 o înregistrare alcătuită din 11 caractere, cu următoarea formă :

- .40979E + 03

Numărul se scrie aliniat la dreapta în cele w poziții. Pozițiile neocupate ($w - d = 6$) se completează cu blancuri.

Descriptorul F. Reprezentarea valorii variabilei se face în w poziții ultimele d conținând partea fracționară. Ea este separată de partea întreagă prin punctul zecimal.

Descriptorul D. Reprezentarea se face normalizat, sub una din formele următoare :

$$\begin{aligned} & .X_1X_2\dots X_dD \pm Y_1Y_2 \\ & -.X_1X_2\dots X_dD \pm Y_1Y_2 \end{aligned}$$

unde X_1, \dots, X_d sînt cifre zecimale, iar Y_1Y_2 constituie partea de exponent, $X_1 \neq 0$.

● *Descriptori pentru transmiterea valorilor variabilelor complexe*

Transmiterea unei valori complexe se face folosind doi descriptori de tipul F, E sau D, unul pentru partea reală și altul pentru partea imaginară.

Exemplu. Dacă într-un program se folosește variabila complexă C pentru a citi valoarea ei de pe o cartelă se poate folosi instrucțiunea :

44 | | FORMAT (F7.3, E11.5)

● **Descriptorul L pentru transmiterea valorilor variabilelor logice**

Forma generală : Lw

Semnificație : În cele w caractere la intrare, valorile logice adevăr și fals se reprezintă prin literele T și respectiv F, scrise în orice poziție a câmpului. La ieșire, reprezentarea valorilor logice se face prin T sau F scrise în ultima poziție a câmpului.

II 2.6.2. Așezarea câmpurilor în înregistrare și a înregistrărilor în cadrul setului de date

Modul în care apar valorile variabilelor în înregistrările de intrare/ieșire nu este indiferent. De multe ori este necesară aranjarea valorilor variabilelor într-un anumit mod, intercalarea unor texte suplimentare care să explice despre ce valori este vorba etc.

Programatorul poate folosi pentru a descrie poziția câmpului în înregistrare și a unei succesiuni de înregistrări, codurile $H''X$, caracterul de control al imprimării și separatorului /. Aceste coduri nu se asociază unor variabile din lista de intrare/ieșire.

Descriptorul X

Forma generală : nX

Semnificație la intrare: ignorarea a n caractere consecutive, *la ieșire*: inserarea a n spații între două zone ale înregistrării¹.

● *Descriptorul de afișare a textelor*

Forma generală: ' $\langle \text{text} \rangle$ '

Semnificație: Acest cod permite ca la ieșire să fie afișat textul indicat între apostroful¹.

Exemplu. Instrucțiunile:

```
22 || X = 2.5  
   || X = 4 * X + 3.236  
   || WRITE (108, 22)X, Y  
   || FORMAT ('||X = ', F9.2, 2X, 'F(X) = ', E12.3)
```

vor produce afișarea la imprimantă a unui text de forma:

||X = |||||2.50||F(X) = |||||.132E+02

Observație: Primul caracter a fost utilizat pentru controlul imprimării

Forma generală: $nHc_1c_2 \dots c_n$

Semnificație: la ieșire este imprimat pe n poziții textul $c_1c_2 \dots c_n$ care urmează după H¹. Are același efect ca și codul precedent.

Exemplu: Instrucțiunea FORMAT din exemplul precedent ar putea avea forma următoare, efectul rămânând neschimbat,

```
122 || WRITE (108, 122)X, Y  
    || FORMAT (4H ||X = ', F9.2, 2X, 5H F(X) = ', F12.3)
```

având însă același efect.

● *Caracterul de control al imprimării*

Un rând scris pe imprimantă poate fi alcătuit din cel mult 132 de caractere. Pentru ca așezarea rândurilor pe pagina de imprimantă să poată fi hotărâtă de programator, s-a convenit ca primul caracter din înregistrarea de ieșire să nu fie imprimat, el dând informațiile necesare în acest sens.

La descrierea unei înregistrări pentru ieșire, primul caracter este caracter de control al imprimării și are una din formele:

- 0 — avans cu două linii înaintea imprimării,
- 1 — avans la prima linie a paginii următoare,
- + — nici un avans.

Blank sau orice caracter diferit de 0, 1 și + înseamnă avans cu o linie înaintea imprimării.

Caracterul de control poate fi specificat în lista FORMAT astfel: 1 H ν sau ' ν ', unde ν are una din cele patru forme menționate anterior.

● *Separatorul / (slash)*

Utilizarea separatorului / (slash) în $\langle \text{listă-descriptor} \rangle$ are următoarele efecte:

Dacă la începutul sau sfârșitul listei FORMAT apar n slash-uri, atunci la intrare vor fi sărite n înregistrări, iar la ieșire n înregistrări vor fi pline cu blankuri.

¹ În cazul în care dispozitivul de ieșire este imprimanta, primul caracter al înregistrării nu se afișează.

Exemplu :

```
33 || REAL A(5)  
    || READ (105, 33) (A(I), I = 1, 5)  
    || FORMAT (///5F7.2)
```

La execuție, vor fi sărite trei cartele (înregistrări), valorile elementelor tabloului fiind citite de pe a 4-a cartelă.

Când apare un slash în interiorul unei liste *FORMAT* se trece la înregistrarea următoare. Prin utilizarea a n slash-uri consecutive în interiorul unei <listă-descriptori> vor fi sărite la intrare $n - 1$ înregistrări, sau se vor introduce $n - 1$ înregistrări cu spații, la ieșire.

Observație — Când slash-ul separă 2 coduri în lista *FORMAT*, nu se mai folosesc paranteze

11.2.7. Reguli de alcătuire a listei *FORMAT*

Lista *FORMAT* este formată din descriptori care descriu structura înregistrării.

În alcătuirea acestei liste pot fi folosite următoarele posibilități :

a) Dacă un descriptor d se repetă de k ori consecutiv atunci poate fi folosită în listă forma echivalentă kd .

Exemplul 1. Instrucțiunea

```
14 || FORMAT (14, 14, 14, F5.2, 14)
```

este echivalentă cu următoarea :

```
14 || FORMAT (314, F5.2, 14)
```

unde 3 indică numărul de repetări al descriptorului 14.

b) Dacă un grup de descriptori d_1, d_2, \dots, d_i se repetă în aceeași ordine de k ori, atunci în descrierea <listă-descriptori> se poate folosi un element de forma $k(d_1, d_2, \dots, d_i)$.

Exemplul 2. Instrucțiunea :

```
51 || FORMAT (I3, I4, E11.5, I4, E11.5, D14.6)
```

este echivalentă cu următoarea :

```
51 || FORMAT (I3, 2(I4, E11.5), D14.6)
```

11.2.8. Reguli după care se explorează lista *FORMAT*

Pentru fiecare variabilă care apare în lista de intrare/ieșire se stabilește care este descriptorul corespunzător din <listă-descriptori> (pe care o vom desemna și sub numele echivalent de listă *FORMAT*), descriptor care precizează reprezentarea externă a valorii variabilei.

Fie n numărul variabilelor și al elementelor de tablou din lista de intrare/ieșire. În lista instrucțiunilor *FORMAT* asociată, presupunem că se află m descriptori pentru transmiterea valorilor variabilelor.

În funcție de valorile lui m și n desprindem următoarele acțiuni:

a) $n = m$ — Lista de intrare/ieșire se poate satisface doar printr-o singură parcurgere a listei *FORMAT*. Fiecărei variabile sau element de tablou îi corespunde un singur descriptor din lista *FORMAT*.

(b) $n < m$ — Lista de intrare/ieșire poate fi satisfăcută fără să folosim toți descriptorii de tratare a variabilelor din lista *FORMAT*. În acest caz sînt folosiți doar primii n descriptori.

Exemplul 1.

```
33 || HEAD (1, 33) I, J, K
    || FORMAT (I6, I7, I5, I4, I2)
```

Descriptorii I4, I2, nu sînt utilizați:

(c) $n > m$ — Foarte variabilele din lista de intrare/ieșire nu pot fi satisfăcute printr-o singură explorare a listei *FORMAT*. Pentru a se citi/scrie valorile tuturor variabilelor, se va relua explorarea formatului citindu-se/scriindu-se o nouă înregistrare;

Explorarea se reia de la început, dacă în listă nu apar grupuri de descriptori închuse între paranteze. În cazul cînd apar astfel de grupuri, explorarea se reia de la ultimul grup folosit. O regulă practică de găsim a acestuia este: se ia grupul a cărui paranteză este ultima închisă în dreapta (parantezele ce închid lista *FORMAT* nu se iau în considerare).

Exemplul 2. Fie instrucțiunile:

```
21 || WRITE (3,20) X, Y, Z, V, W, T
    || FORMAT (2F2.1, 3F7.2)
```

Explorarea se reia de la început. În prima înregistrare vor fi scrise valorile variabilelor X, Y, Z, V, W iar în a doua, doar valoarea lui T descriptorul fiind F2.1.

Dăm mai jos alte forme ale instrucțiunii, precum și punctele din care se reia explorarea, pentru aceeași listă de variabile.

```
21 || FORMAT (2F2.1, 3(F7.2))
```

↑
reluarea explorării

```
21 || FORMAT (F2.1, (F2.1, 3F7.2))
```

↑
reluarea explorării

```
21 || FORMAT (F2.1, (F2.1, F7.2), 2F7.2)
```

↑
reluarea explorării

11.2.9. Exemple de programe FORTRAN

Exemplul 1 Programul de rezolvare a ecuației de gradul 1, din figura 11.2, poate fi acum înțeles pe deplin. Instrucțiunile cu numerele 130 până la 160 descriu structura înregistrărilor

```

1      INTEGER R, T
      READ (105, 1) M, N
      FORMAT (2I6)
      IF (M GE N) GOTO 100
      T = N
      N = M
      M = T
100     CONTINUE
101     R = M - (M/N)*N
      M = N
      N = R
      IF (R NE 0) GOTO 101
      WRITE (108, 2) M
      FORMAT (1, 'C M M D C ', 16)
      STOP
      END
2

```

Fig. 11.12

de intrare și de ieșire. Căutați să lămurii rolul fiecărei părți a acestor instrucțiuni.

Exemplul 2 Algoritmul lui Euclid (programul LPS, cap. 13.2) Programul corespunzător în limbajul FORTRAN este cel din figura 11.12: în acest program valorile variabilelor M și N sînt citite de pe o cartelă, T și R sînt declarate de tip întreg. Analizați programul FORTRAN insistînd în mod deosebit asupra instrucțiunii cu eticheta 101.

Din exemplele de programare de mai sus se remarcă modul în care au fost codificate în FORTRAN instrucțiunile limbajului LPS. În figura 11.13 este arătată schema după care recomandăm să se codifice instrucțiunea condițională, iar în figura 11.14 schemele

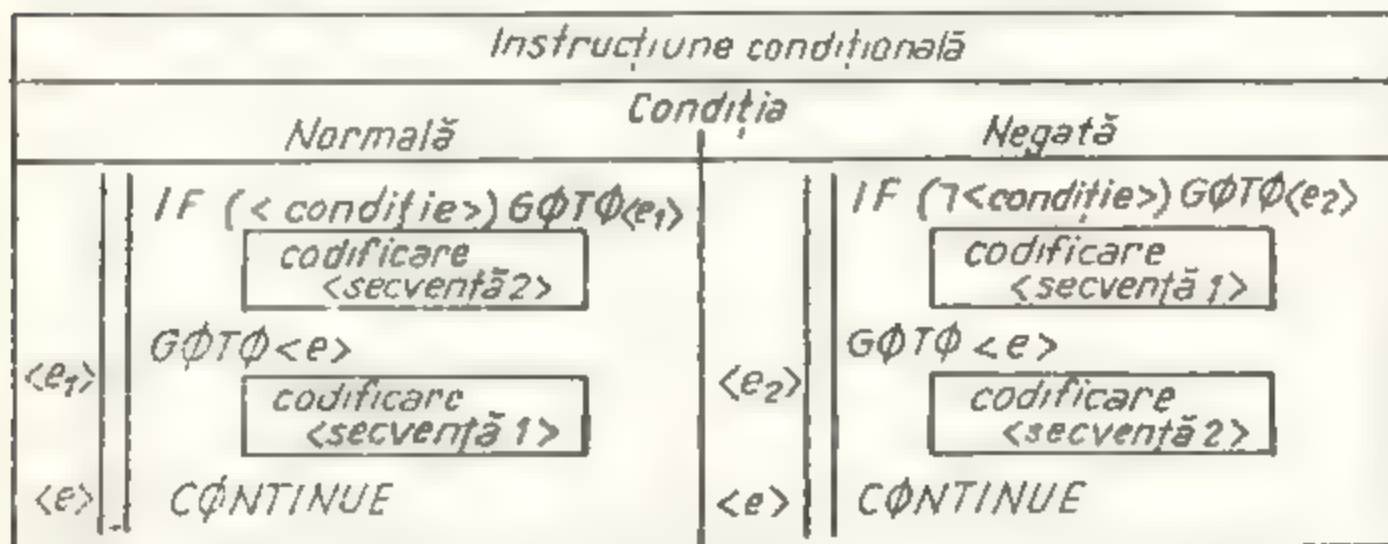


Fig. 11.13

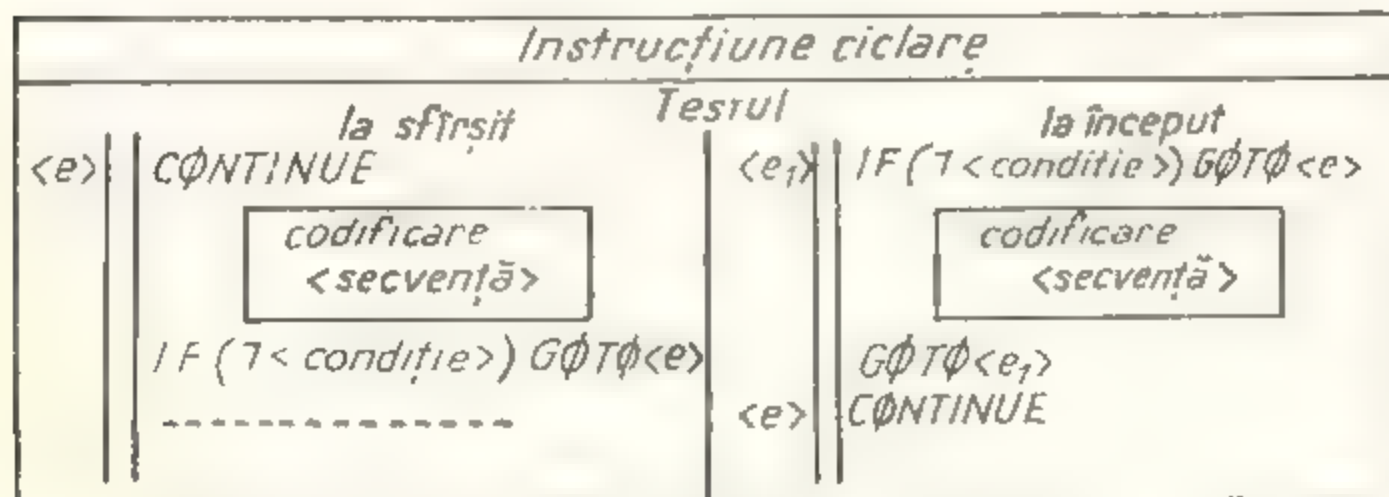


Fig. 11.14

```

1  INTEGER A(1000), T
   LOGICAL SEM
   READ (105, 1) N
1  FORMAT (I3)
   READ (105,2) (A(K), K = 1, N)
2  FORMAT (8110)
100 CONTINUE
    SEM = .TRUE.
    I = 1
200 IF(I GT N-1) GO TO 201
    IF(A(I) GE A(I + 1))GO TO 301
    T = A(I)
    A(I) = A(I + 1)
    A(I + 1) = T
    SEM = .FALSE.
301 CONTINUE
    I = I + 1
    GO TO 200
201 CONTINUE
   IF (.NOT.SEM) GO TO 100
   WRITE (108, 3) (A(I), I = 1, N)
3  FORMAT (' 30, 'TABLOUL A ORDONAT' (10112))
   STOP
   END

```

Fig. II.15

cu excepțiile respective ale instrucțiunilor de control: ca test inițial și cu text final. În aceste figuri s-au respectat notațiile din LPS iar etichetele FORTRAN sînt notate cu $\langle e_1 \rangle$, $\langle e_2 \rangle$ și $\langle e \rangle$. Precizăm că ori de cîte ori instrucțiunea CONTINUE nu este necesară, ea nu va fi folosită.

Exemplul 3. Ordinea unui fir de numere (algoritmul LPS, cap. I 3.3). Programul este dat în figura II.15. Au fost folosite schemele de codificare recomandate mai sus. De asemenea, precizăm că programul poate ordona și ori de cît mai mult 999 de numere, perforate pe cartele, numărul acestora, N, fiind perforat în primele trei coloane ale primei cartele.

Exemplul 4. Rădăcinile ecuației de gradul al doilea $ax^2 + bx + c = 0$ unde $a, b, c \in \mathbb{R}$ pot fi găsite cu ajutorul unui program FORTRAN. Condițiile care trebuie luate în considerare în alcătuirea programului sînt:

- valorile coeficienților sînt citite de pe o cartelă, coeficienții vor fi scriși cu patet zecimal fiecare ocupînd cîte zece coloane:

- în funcție de valorile coeficienților a, b, c ecuația va avea două soluții, o soluție, o infinitate sau nici una, după cum $(a \neq 0)$, $(a = 0$ și $b \neq 0)$, $(a = b = c = 0)$ și respectiv $(a = b = 0$ și $c \neq 0)$, pe imprimantă se cere să se afișeze ce ecuație s-a rezolvat și rădăcinile acesteia, în cazul în care acestea există.

Descrierea algoritmului în LPS este dată în continuare.

citește a, b, c

<p>dacă $a \neq 0$ atunci</p>	<p>dacă $b \neq 0$ atunci</p>	<p>dacă $c = 0$ atunci scrie 'ecuație nedeterminată' scrie 'soluție este orice $x \in \mathbf{R}$' altfel scrie 'ecuație imposibilă'</p>
	<p>altfel atribuie $x \leftarrow -c/b$ scrie 'ecuație gradul I', a, b, c, x</p>	
	<p>altfel atribuie $d \leftarrow b^2 - 4ac$ dacă $d > 0$ atunci atribuie $x_1 \leftarrow (-b + \sqrt{d})/(2a)$ & $x_2 \leftarrow (-b - \sqrt{d})/(2a)$ scrie 'ecuație gradul II, rădăcini reale' scrie a, b, c, x_1, x_2</p>	
	<p>altfel atribuie $d \leftarrow -d$ & $x_{real} \leftarrow -b/(2a)$ & $x_{imag} \leftarrow \sqrt{d}/(2a)$ scrie 'ecuație gradul II, rădăcini complexe'</p>	
	<p>scrie $a, b, c, x_{real}, x_{imag}$.</p>	

Programul FORTRAN este :

C	PROGRAMUL REZOLVĂ O ECUAȚIE DE TIPUL
C	$A \cdot X^2 + B \cdot X + C = 0$
C	UNDE A, B, C, SÎNT CITIȚI DE PE CARTELE
	READ (105, 100) A, B, C
	IF (A.NE.O) GO TO 200
	IF (B.NE.O) GO TO 21
	IF (C.NE.O) GO TO 11
	WRITE (108, 101)
	GO TO 99
11	WRITE (108, 102)
	GO TO 99
21	X = -C/B
	WRITE (108, 103) A, B, C, X
	GO TO 99


```

200      DELTA = B**2 - 4*A*C
      IF(DELTA.LT.0) GOTO 211
      X1 = (-B + SQRT (DELTA))/(2*A)
      X2 = (-B - SQRT (DELTA))/(2*A)
      WRITE (108, 104) A, B, C, X1, X2
      GOTO 99
211      XREAL = -B/(2*A)
      XIMAG = SQRT (-DELTA)/(2*A)
      WRITE (108, 105) A, B, C, XREAL, XIMAG
      GOTO 99
99      STOP
100      FORMAT (3 F10.3)
101      FORMAT ('□', 'ECUAȚIE NEDETERMINATĂ' /
+      '□ORICE X DIN R ESTE SOLUȚIE')
102      FORMAT ('□ECUAȚIE IMPOSIBILĂ')
103      FORMAT ('□ECUAȚIE DE GRADUL I', 4F10.3)
104      FORMAT ('□ECUAȚIE DE GRADUL DØI, RĂDĂCINI
+      REALE' / '□', 3F10.3/'□', 2F10.3)
105      FORMAT ('□ECUAȚIE DE GRADUL DØI, RĂDĂCINI'
+      '□COMPLEXE' / '□', 3 F10.3/'□', 2F10.3)
      END

```

11.2.10. Exerciții

4. Scrieți în limbajul FORTRAN următoarele fraze:

- Dacă $x \geq b$ valoarea lui x e egală cu 15.6, altfel e egală cu 21.5
- Dacă $\alpha + \beta$ e pozitiv atunci $y = 15.7$ altfel dacă $x < 7$ atunci $y = -21$ altfel $y = 121.1$
- Dacă $\alpha y - 5$ e egală cu zero execută în continuare instrucțiunea cu eticheta 129, altfel execută instrucțiunea următoare, cu numărul 767

5. Scrieți în FORTRAN:

- Dacă $a < 0$ și $b > 0$ sau dacă $a = 0$ atunci $x = -2$.
În caz contrar valoarea lui x nu se schimbă.
- Dacă $10^{-7} \leq x \leq 1$, se execută instrucțiunea cu eticheta 777, altfel se oprește procesul de calcul (STOP).

6. Se dă valoarea lui K. Să se descrie în limbajul FORTRAN următoarea prelucrare: Dacă valoarea lui K este pară se execută instrucțiunea cu numărul 615, dacă e impară se continuă prelucrarea.

7. Variabila RAD conține valoarea unui unghi exprimată în radiani. Alcătuiți schema logică și programați după a căror execuție variabilele întregi GR, MIN și SEC, conțin numărul de grade, minute și, respectiv, secunde corespunzătoare unghiului RAD.

8. Care e rezultatul lui S după executarea următoarelor programe:

```

a)
1  I = 1
6  P = 7.1
   IF(I-4)2, 3, 2
2  I = I + 1
   GO TO 6
3  S = P
   WRITE (108, 1)S
1  FORMAT ('□', F7.2)
   STOP
   END

```

```

b)
   S = 7.5
7  IF(S - 10)5, 6, 6
5  S = S*1.2
   GO TO 7
6  S = S + 1
   WRITE (108, 1)S
1  FORMAT ('□', F7.2)
   STOP
   END

```

6. Identificați erorile comise în scrierea următoarelor instrucțiuni :

```

      GØ TØ K
      IF(K - 5) 6, 6, 67.
      V + 5 = 2 + IVAR
      IF(K*5) - 6) 5.2, 3
      I + 7 = ALFA
      IF(VAX)Ø, 6, 3, 2
      1 VALOARE = ABCDEIFGH
  
```

7. Scrieți programul în limbajul FORTRAN care să citească de pe o cartelă valoarea variabilei N și să afișeze apoi valoarea variabilei S unde

$$S = 1 + 2 + \dots + n$$

Indicație. Calculați pe S fără să folosiți formula $S = \frac{n(n+1)}{2}$.

8. Scrieți un program în limbajul FORTRAN care să găsească numerele prime între 1 și N unde N este citit de pe o cartelă.

9. Indicați toate variabilele și elementele de tablou care fac parte din listele instrucțiunilor READ/WRITE, de mai jos :

```

      REAL A(5), B(5 7), C(49)
      INTEGER M (17)
      DOUBLE PRECISION F (10, 20), D(4, 2, 5)
      COMPLEX C (21)
      READ (105, 78) (A(I), I = 1, 5, 2)
      READ (105, 79) A(2), ((B(L, M), M = 1, 7 3), L = 1, 4)
      READ (105, 80) A(6), (B(2, 4), (C(IV), IV = 4, 21, 4)
      WRITE (108, 21) (((D(I, J, K), J = 1, 2), I = 1, 4), K = 1, 5, 2)
      WRITE (108, 22) A, ((B(I, J), J = 1, 3), I = 1, 5)
      WRITE (108, 23) ((F(L, M), L = 4, 8), M = 5, 7), (C(1), C(7))
  
```

11.3. FACILITĂȚI DE PROGRAMARE ÎN LIMBAJUL FORTRAN

Cu instrucțiunile studiate până în prezent se poate descrie în FORTRAN aproape orice prelucrare. Totuși limbajul oferă multe posibilități de programare, importante, dintre care o parte o vom studia în continuare : instrucțiunea DO și subprogramele.

11.3.1. Instrucțiuni DO

Această instrucțiune este replica în FORTRAN a instrucțiunii de ciclare cu contor din limbajul LPS. Datorită asemănării lor vom da mai puține explicații asupra prelucrărilor exprimate de aceasta.

Formă generală :

DO n contor = vinit, vfin, vpas

unde :

- n este eticheta unei instrucțiuni executabile ce urmează în program după instrucțiunea DO ;
- contor se numește variabila ciclului și poate fi un nume de variabilă întregă (nu poate fi element de tablou) ;
- vinit, vfin, vpas se numesc parametrii ciclului ; ei sînt fie variabile de tip întreg cu valori pozitive (nu elemente de tablou), fie constante pozitive fără semn.

În cazul cînd valoarea lui vpas este 1, instrucțiunea se scrie astfel :

DO n contor = vinit, vfin

Este necesar să avem $0 < v_{\text{init}} \leq v_{\text{fin}}$ și $v_{\text{pas}} > 0$.

Semnificație: Executarea acestei instrucțiuni începe prin a atribui variabilei *contor* valoarea lui *v_{init}* după care:

se execută toate instrucțiunile care urmează după DØ pînă la instrucțiunea cu eticheta *m* inclusiv (acest grup de instrucțiuni se numește uneori corpul ciclului DØ),

valoarea variabilei *contor* este mărită cu aceea a lui *v_{pas}* (cînd *v_{pas}* lipsește, cu 1);

- dacă valoarea lui *contor* depășește pe aceea a lui *v_{fin}* atunci se continuă execuția programului cu instrucțiunea care urmează imediat după aceea cu eticheta *n*, altfel se vor aplica din nou regulile de mai sus.

Exemplu. Înmulțirea a două matrice. În FORTRAN programul va fi următorul:

	INTEGER P
	REAL A (20, 15), B(15, 10), C(20, 10)
C****	CITIREA DIMENSIUNILOR REALE ȘI A MATRICELOR**
	READ (105,1) N, M, P
	READ (105,2) ((A(I, J), I = 1, N), J = 1, M)
	READ (105,2) ((B(I, J), I = 1, M), J = 1, P)
C****	ÎNMULȚIREA MATRICELOR
	DØ 11 I = 1, N
	DØ 12 K = 1, P
	S = 0
	DØ 13 J = 1, M
	S = S + A(I, J)*B(J, K)
13	CONTINUE
	C(I, K) = S
12	CONTINUE
11	CONTINUE
C****	AFIȘARE REZULTATE
	WRITE (108,3) ((C(I, J), J = 1, P), I = 1, N)
	STOP
1	FORMAT (3I5)
2	FORMAT (16 F5.1)
3	FORMAT ('□', 10 F8.2)
	END

Observații:

1. Dimensiunile reale ale matricelor sînt citite la începutul programului. Asupra mărimilor nu se face în program nici o verificare, ele fiind presupuse corect transmise. În caz contrar rezultatul execuției programului ar fi eronat.
2. Dacă dimensiunile reale ale matricelor coincid cu cele din declarațiile corespunzătoare atunci listele de intrare/ieșire sînt mult mai simple:

```

READ (105, 1) A
READ (105, 1) B
WRITE (108, 3) C

```

3. Pe imprimantă identificarea liniilor matricei C, dacă numărul exact al coloanelor este diferit de 10, va fi dificilă. O formă îmbunătățită de afișare ar fi fost obținută astfel:

```

22 || DØ 22 I = 1, N
   || WRITE (108, 3) (C(I, J), J = 1, P)
   || CONTINUE

```

Dintre regulile care trebuie respectate cînd folosim această instrucțiune, le amintim pe următoarele :

— Valorile variabilelor *count*, *unit*, *v/in*, sau *xpas* nu pot fi schimbate prin instrucțiunile din corpul ciclului ;



Situatii permise

Situatii interzise

Fig. II.16

— Se poate sari din interiorul corpului în afara acestuia, dar nu și invers (vezi figura II.16). Saltul la o instrucțiune din interiorul corpului ciclului este permis numai dacă anterior s-a executat un salt din interiorul acestuia și nu s-au modificat parametrii ciclului.

Ultima instrucțiune a corpului unui ciclu **DO** nu poate fi : **GO TO**, **IF** aritmetic, **PAUSE**, **STOP**, **RETURN**, **DO**, sau **IF** logic în care apare una din instrucțiunile de mai sus.

II.3.2. Subprograme

Cînd este necesară îndeplinirea de mai multe ori într-un program a aceluiași calcul pentru date diferite, se poate folosi conceptul de *subprogram*. În subprogram, care poate fi privit ca o unitate independentă, calculele ce se repetă sînt descrise o singură dată, iar din altă unitate de program se poate cere efectuarea acestora ori de cîte ori este nevoie.

Alte subprogramele cit și programul principal constituie părți componente ale aceluiași program. Totuși ele sînt independente, în sensul că fiecare dintre acestea poate fi alcătuit separat de celelalte, poate folosi nume de variabile care au mai fost întrebuițate și în restul programului și poate fi compilat și testat separat.

Acste caracteristici permit ca la rezolvarea unei probleme să poată lucra mai multe persoane, fiecare elaborînd cîte o parte din program. Unirea acestor părți pentru a rezolva problema considerată se realizează numai atunci cînd fiecare din ele este corectă din punct de vedere sintactic.

Important pentru cîl ce folosește subprogramul este să știe care sînt datele de intrare și unde va fi găsit rezultatul. La orice centru de calcul există o așa-numită „bibliotecă”, ce cuprinde asemenea programe.

În continuare vom numi **unitate de program** fie programul principal, fie un subprogram. O unitate de program este o parte de sintaxă a unui program, independentă de celelalte unități de program (în sensul prezentat mai sus).

Orice unitate de program se alcătuește după regulile corespunzătoare limbajului. Pîna acum toate programele pe care le-am scris au fost programe principale.

Ordinea instrucțiunilor FORTRAN într-o unitate de program este următoarea :

1. O instrucțiune SUBROUTINE sau FUNCTION, dacă există
2. Instrucțiunea IMPLICIT, dacă e cazul.
3. Declarații asupra datelor folosite în unitatea de program.
4. Cel puțin o instrucțiune executabilă.
5. Instrucțiunea END.

Dacă în unitatea de program apar instrucțiunile de la punctul 1 avem de-a face cu un program, în caz contrar, cu un *program principal*.

Dez din punct de vedere al formei, un subprogram este un grup de instrucțiuni FORTRAN a căror primă instrucțiune este SUBROUTINE sau FUNCTION, iar ultima este END.

În mod similar, un program principal e alcătuit dintr-un grup de instrucțiuni FORTRAN printre care nu figurează SUBROUTINE, FUNCTION, dar a cărei ultimă instrucțiune este END.

Din punct de vedere al utilizării, subprogramele se deosebesc prin :

- a) modul de alcătuire al unității de program respective ;
- b) modul de referire (chemare), adică modul cum se exprimă într-o unitate de program necesitatea prelucrării efectuate de un subprogram ;
- c) modul de punere la dispoziția unității de program care cheamă subprogramul, a rezultatilor prelucrării efectuate de subprogram.

Pe baza acestor caracteristici subprogramele se împart în :

— *subprograme de tip funcție* (numite și funcții externe) a căror primă instrucțiune este FUNCTION ;

— *subprograme de tip subrutină* (mai scurt, subrutine) a căror primă instrucțiune este SUBROUTINE.

II.3.2.1. Funcții externe

Dacă prelucrarea cerută de găsirea valorii unei funcții nu se poate exprima printr-o singură expresie aritmetică, iar valoarea acestei expresii este necesară și în altă unitate de program, atunci pentru definirea funcției va fi alcătuită o unitate separată de program. În alte

unități de program se pot face referințe la funcția astfel definită, având drept urmare calculul valorii funcției pentru diferite valori ale argumentelor ei.

Structura generală a unei funcții externe este cea din figura II.17.

În definirea funcțiilor externe apar două instrucțiuni noi, FUNCTION și RETURN.

<i>tip FUNCTION (argumente fictive)</i>
— Declarații
— Instrucțiuni executabile (cel puțin una)
— Instrucțiunea RETURN
END

Fig. II.17

Instrucțiunea FUNCTION

Forma generală :

|| t FUNCTION f(a₁, a₂, ..., a_n)

unde : *t* - Tipul numelui funcției (INTEGER, REAL, DOUBLE PRECISION, COMPLEX, LOGICAL) Dacă este omis, tipul numelui funcției se stabilește ca pentru o variabilă FORTRAN oarecare.

f - Numele funcției. El va păstra valoarea funcției

a₁, a₂, ..., a_n - Lista argumentelor fictive. Ea trebuie să conțină cel puțin un element. Elementele listei trebuie să fie distincte și pot fi nume de variabile sau nume de tablouri, dar nu elemente ale unor tablouri.

Semnificație Definește numele unei funcții, tipul și argumentele fictive ale acesteia.

Exemplu :

```
REAL FUNCTION MARE (PAR)  
COMPLEX FUNCTION CRIC (A, B)  
FUNCTION MIC (X, Y, Z).
```

În exemplul al treilea funcția MIC va fi de tip întreg, dacă o declarație IMPLICIT nu indică alt tip

Instrucțiunea RETURN

Forma generală :

|| RETURN

Semnificație . Produce transferul controlului, în programul care a cerut execuția funcției externe.

Într-o funcție externă pot exista mai multe instrucțiuni RETURN

La definiția unei funcții externe trebuie respectate următoarele reguli :

R1 Se va stabili care e prelucrarea îndeplinită de funcție, numele funcției și al argumentelor sale.

R2 Nu se pot folosi instrucțiunile SUBROUTINE sau o altă instrucțiune FUNCTION.

R3. Numelui funcției trebuie să-i fie atribuită o valoare în cadrul subprogramului.

Exemplul 1

Într-un program se repetă de mai multe ori calculul funcției :

$$y = \begin{cases} \frac{x}{1 + e^{\frac{1}{x}}} & \text{dacă } x < -2, \\ e^{-x^2} & \text{dacă } -2 \leq x < 4, \\ x^2 \sin \frac{1}{x} & \text{dacă } 4 \leq x \end{cases}$$

S-a convenit să se alcătuiască funcția externă YFUN folosind drept argument fictiv variabila X.

Respectind regulele de mai sus, obținem subprogramul :

```
31 REAL FUNCTION YFUN(X)
    IF(X GE - 2.0) GO TO 31
    YFUN = X/(1 + EXP(1/X))
    RETURN
32 IF(X GE 4) GO TO 32
    YFUN = EXP(-(X*X))
    RETURN
33 YFUN = X**2*SIN(1/X)
    RETURN
END
```

Valoarea funcției va fi păstrată de YFUN, numele funcției. Se observă întrebuintarea a trei instrucțiuni RETURN.

Folosirea funcțiilor externe

Pentru utilizarea corectă a unei funcții externe, se va pune seama de următoarele reguli :

S1. Se vor stabili care sînt argumentele efective pentru care se va calcula valoarea funcției.

S2. Argumentelor efective trebuie să le corespundă ca număr, ordine și tip argumentele fictive.

— Dacă argumentul fictiv corespunzător este un nume de variabilă atunci argumentul efectiv poate fi orice expresie de același tip.

Dacă argumentul fictiv corespunzător este un tablou, trebuie ca argumentul efectiv corespunzător să fie tot un nume de tablou. În plus, pentru a evita orice erori este bine ca tablourile corespunzătoare să aibă aceleași dimensiuni.

S3. Găsirea valorii unei funcții externe pentru argumentele efective stabilite este cerută scriind numele funcției urmat între paranteze de lista argumentelor efective.

S4. Cînd este întâlnită o referire la o funcție sînt îndeplinite următoarele acțiuni :

— Se găsește valoarea fiecărui argument efectiv.

— Folosind aceste valori drept valori ale argumentelor fizice corespundente, se execută prelucrarea indicată de funcția externă.

— La întâlnirea primei instrucțiuni RETURN se revine în locul unde a fost făcută referirea la funcție. În acest moment la locul referirii se va pune valoarea numelui funcției, continuîndu-se execuția programului.

Exemplul 2 : Fie o referire la funcția YFUN (din exemplul 1) de forma :

$$Z = 2.4 + 5 * YFUN(0.0).$$

După executarea acestei instrucțiuni, Z va avea valoarea 7.4 (adică $2.4 + 5$).

Tablouri din subprograme ale căror dimensiuni sînt precizate în momentul execuției.

Funcțiile externe sînt utile în multe aplicații cu tablouri. Totuși de multe ori nu se cunosc dimensiunile exacte ale tablourilor ce sînt folosite, ci numai cîte dimensiuni au ele.

În subprograme (și numai în ele) se poate utiliza o formă particulară a declarației DIMENSION în care nu se indică mărimea dimensiunilor unui tablou prin constante, ci prin variabile întregi, care sînt parametri ai subprogramului. Mărimile acestor dimensiuni vor fi precizate în momentul execuției.

Exemplul 3. Să se alcătuiască programul de tip funcție, PRØD, care să calculeze $\sum_{i=1}^n a_i b_i$, unde a_i și b_i sînt componentele a două tablouri reale cu cîte n elemente.
Drept argumente fictive vom folosi:
 A, B , — tablouri cu n elemente;
 N — variabila a cărei valoare va indica numărul elementelor tablourilor

```

77 | FUNCTION PRØD (A, B, N)
    | DIMENSION A(N), B(N)
    | PRØD = 0
    | DO 77 K = 1, N
    |     PRØD = PRØD + A(K) * B(K)
    | CONTINUE
    | RETURN
    | END

```

Iată și o referire la această funcție externă:

```

1 | REAL X(200), Y(200)
  | . . . . .
  | . . . . .
  | CALCUL = PRØD (X, Y, 200) + 5.72
  | . . . . .

```

Cînd se întîlneste PRØD (X, Y, 200), la evaluarea funcției se consideră că dimensiunea celor două tablouri este 200 (valoarea argumentului efectiv corespunzător lui N).

II.3.2.2. Subrutine

Ca și funcțiile externe, subrutinele constituie unități distincte de program. Pentru alcătuirea subrutinelor sînt folosite cîteva instrucțiuni specifice, care vor fi studiate în continuare.

Instrucțiunea SUBROUTINE

Servește la definirea subrutinelor, fiind prima instrucțiune a lor.

Forma generală:

| | SUBROUTINE s(a_1, a_2, \dots, a_n)

unde:

s — Numele subrutinei (se formează în mod similar cu numele funcțiilor)
 a_1, a_2, \dots, a_n — Lista argumentelor fictive. Ea poate fi și vidă (în acest caz parantezele nu se mai scriu). Elementele listei sînt nume de variabilă sau de tablou (nu elemente de tablou).

Semnificație: Indică numele subrutinei, numărul, ordinea și tipul argumentelor. Spre deosebire de funcțiile externe, numelui subrutinei nu i se asociază un anumit tip. El servește numai pentru id. nificarea acestora.

Instrucțiunea CALL

Forma generală:

| | CALL s(p_1, p_2, \dots, p_n)

unde:

s — Numele subrutinei;
 p_1, p_2, \dots, p_n — Lista argumentelor efective.

Semnificație: Prin intermediul acestei instrucțiuni se cere execuția subrutinei al cărei nume este s . Argumentele efective sînt indicate în listă și ele trebuie să corespundă ca număr, ordine și tip cu argumentele fictive din definiția subrutinei.

Exemplu : O secvență de apelare a subrutinei NPR într-un program principal este :

```

1  || REAL A(20, 10), B(10, 30), C(20, 30)
   || READ (105, 1) A, B
   || FORMAT (10F8.5)
   || CALL NPR (A, B, C, 20, 10, 30)

```

În acest moment matricea C va conține produsul A-B.

11.3.3. Exerciții

1. În figura 11.18 este reprezentată structura unor cicluri DO, arătați care din salturile indicate sînt permise și care nu. De ce ?

2. Identificați instrucțiunile corecte. De ce sînt greșite celelalte ?

- a) DO 55 K = 1, 2, 3
- b) DO 38 VAX = K(L), K(2), K(3)
- c) DO 54 L(1) = 4, 5
- d) DO K = 1, 12, 4
- e) DO 77 K = 1, 4, 5, 6
- f) DO 88 1, 2, 3

3. Sînt corecte următoarele secvențe ? Să se justifice.

a)

```

77 || DO 77 J = 1, 100
   || PLOG = PLOG + ( - 1)**J, J

```

b)

```

33 || S = 0
   || DO 33 K = 1, 10
   || S = S + K

```

c)

```

82 || DO 88 L = K, L, M
   || S(L) = 3*(M - L)
   || GO TO 60

```

d)

```

11 || DO 11 K = 1, 100
   || A = 5

```

e)

```

22 || S = 0
   || DO 22 I = 1, M
   || DO 22 J = 1, N
   || S = S + A(I, J)

```

f)

```

55 || DO 55 K = 1, 11, 3
   || A(K) = B(K)
44 || DO 44 L = 1, 31
   || B(L) = 0

```

g)

```

66 || S = 0
   || DO 66 K = 1, M
   || S = S + A(K)
   || A(K) = 10

```

h)

```

100 || S = 0
   || DO 9 K = 1, 55
99 || DO 9 L = 1, 63
   || S = S + A(K, L)
   || GO TO 100

```

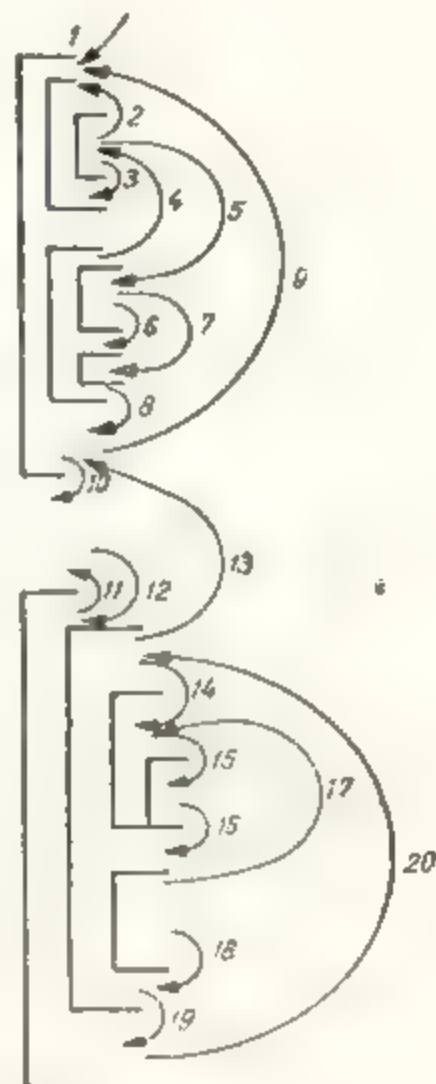


Fig. 11.18

Exemplu. O secvență de apelare a subrutinei NPR într-un program principal este:

```

1  || REAL A(20, 10), B(10, 30), C(20, 30)
   || READ (105, 1) A, B
   || FORMAT (10F8.5)
   || CALL NPR (A, B, C, 20, 10, 30)

```

În acest moment matricea C va conține produsul A·B.

11.3.3. Exerciții

1. În figura 11.18 este reprezentată structura unor cicluri DO, arătați care din salturile indicate sînt permise și care nu. De ce?

2. Identificați instrucțiunile corecte. De ce sînt greșite celelalte?

- a) DO 55 K = 1, 2, 3
- b) DO 38 VAX = K(1), K(2), K(3)
- c) DO 54 L(1) = 4, 5
- d) DO K = 1, 32, 4
- e) DO 77 K = 1, 4, 5, 6
- f) DO 88 1, 2, 3

3. Sînt corecte următoarele secvențe? Să se justifice.

a)

```

77 || DO 77 J = 1, 100
   || PL0G = PL0G - (-1)**J/J

```

b)

```

33 || S = 0
   || DO 33 K = 1, 10
   || S = S + K

```

c)

```

82 || DO 88 L = K, L, M
   || S(L) = 3*(M - L)
   || GO TO 80

```

d)

```

11 || DO 11 K = 1, 100
   || A = 5

```

e)

```

22 || S = 0
   || DO 22 I = 1, M
   || DO 22 J = 1, N
   || S = S + A(I, J)

```

f)

```

55 || DO 55 K = 1, 11, 3
44 || A(K) = B(K)
   || DO 44 L = 1, 31
   || B(L) = 0

```

g)

```

66 || S = 0
   || DO 66 K = 1, M
   || S = S + A(K)
   || A(K) = 10

```

h)

```

100 || S = 1
99 || DO 99 K = 1, 55
   || DO 99 L = 1, 63
   || S = S + A(K, L)
   || GO TO 100

```

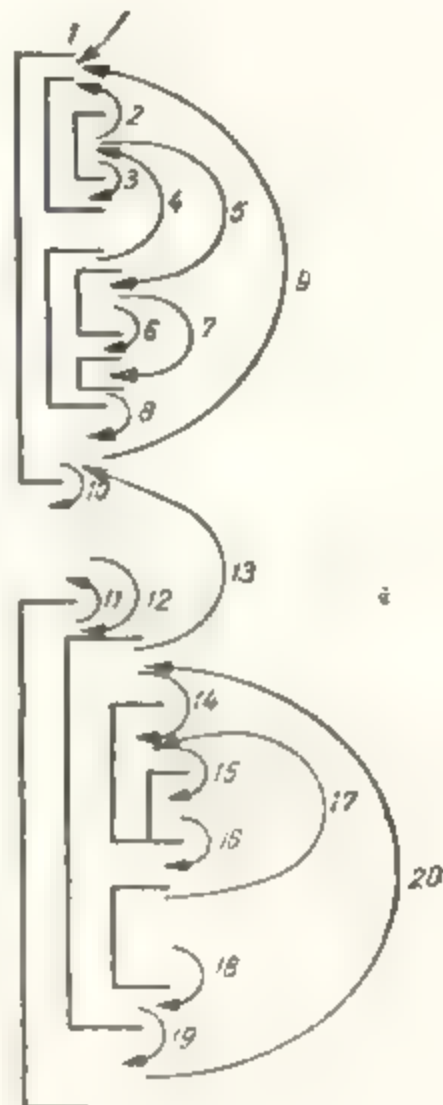


Fig. 11.18

4. Să se scrie un subprogram care să găsească numerele prime cuprinse între 1 și N .
5. Scrieți un subprogram pentru găsirea numerelor perfecte mai mici ca N . (Un număr perfect este un număr care este egal cu suma divizorilor lui, excluzându-se el însuși: $6 = 1 + 2 + 3$; $28 = 1 + 2 + 4 + 7 + 14$)

6. Să se scrie un program care să imprime primii 20 termeni ai șirului lui Fibonacci

$$a_1 = 1, a_2 = 1, \text{ iar } a_i = a_{i-2} + a_{i-1}$$

7. Scrieți un program care să citească de pe cartele valorile a două tablouri A și B unidimensionale, fiecare având câte 25 elemente. Folosind elementele celor două tablouri se completează elementele unui al treilea tablou E de aceeași dimensiune, după regula

$$AE_i = A_i \cdot B_i$$

Să se imprime valorile celor 3 tablouri și valoarea calculată după formula

$$\frac{\sum_{i=1}^{25} A_i \cdot B_i}{25}$$

8. Să se scrie un program care să calculeze

$$\sum_{i=1}^{100} \sqrt{|a_i - b_i| \cdot |a_i + b_i|}$$

unde a_i și b_i sînt elementele a doi vectori reali A și B cu câte 100 de elemente

9. Să se scrie un subprogram FORTRAN care să găsească toate numerele întregi x și y mulți în a , în valoare absolută decât un N fixat, care sînt soluții ale ecuației

$$a) 3x + 5y = 11; \quad b) 3x + 2y = 5.$$

10. Să se scrie un program care să alcătuiască tabla de adunare în baza 8.
11. Să se scrie un program care să calculeze $1!, 2!, 3!, \dots$ în momentul cînd $K!$ va depăși un număr întreg N fixat să nu se mai calculeze $(K+1)!$ și o variabilă V să primească valoarea 9999.
12. Se dau două tablouri A și B fiecare avînd 30 elemente. Se cere să se alcătuiască un program în care să fie completate elementele tabloului X cu 30 elemente, astfel

$$X_i = \begin{cases} -1 & \text{dacă } a_i < b_i, \\ 0 & \text{dacă } a_i = b_i, \text{ pentru } i = 1, \dots, 30, \\ +1 & \text{dacă } a_i > b_i. \end{cases}$$

13. Pe o cardă sînt imprimate 50 de cifre zecimale. Să se scrie un program care să găsească numărul cîtoror pîrți, diferite de zero, al celor impare precum și al celor egale cu zero.
14. Pe cartele sînt imprimate numere întregi fără semn, diferite de zero, fiecare avînd cel mult 3 cifre. Să se alcătuiască un program care să afișeze secvențele bucluite (o secvență este un grup de cel puțin 2 numere consecutive) și numărul acestora.

Observație. O secvență va fi afișată indicînd valoarea inițială a numărului de elemente al secvenței.

15. Cele 90 elemente reale ale unui tablou V sînt citite de pe cartele cu formatul 5F8.3. Să se alcătuiască un program astfel ca variabila K să indice cel mai mic număr de elemente ale tabloului pe care trebuie să le adunăm pentru ca suma să depășească numărul 1000. Dacă suma celor 90 de elemente nu depășește valoarea 1000 atunci K va primi valoarea -1 .
16. Să se alcătuiască funcția PVECT care va da produsul primelor k elemente ale vectorului A cu n componente.
17. Fie tabloul A cu N elemente. Construiți funcția externă PART care calculează suma elementelor tabloului pentru care $|A(I)| > R$, unde valoarea lui R e precizată.

18. Să se alcătuiască funcția MEDIE care va calcula pentru un vector real dat A cu N elemente și un K întreg fixat ($K \leq N$) expresia

$$\frac{A(1) + \dots + A(K)}{K}.$$

19. Să se alcătuiască o funcție care să verifice dacă un tablou A cu N linii și N coloane este simetric (Indicație: Rezultatul se va comunica prin valori diferite ale funcției pentru cazurile posibile).
20. Să se construiască o funcție care să calculeze pentru un tablou X cu M linii și N coloane suma

$$\sum_{i=1}^M \sum_{j=1}^N |x_{ij}|^2.$$

21. Scrieți o funcție care să găsească rădăcina cea mai mare în valoare absolută a unei ecuații de gradul doi.
22. Să se alcătuiască o subrutină care pentru un tablou A cu 15 linii și 30 coloane să calculeze

$$\frac{\sum_{i=1}^{15} \sum_{j=1}^{30} A(i, j)}{450}.$$

23. Fără dat tabloul întreg TAB cu M linii și N coloane alcătuiți o subrutină care să calculeze:

- suma elementelor pozitive;
- suma elementelor negative;
- numărul elementelor egale cu zero.

24. Este posibil ca într-o subrutină cu un singur argument, să se transmită dintr-un program o valoare cu care să se efectueze prelucrarea și apoi să se aducă în programul inițial rezultatul prelucrării?

25. Precizați care sînt deosebirile dintre funcțiile externe și subrutine.

26. Să se construiască o funcție care să arate numărul de apariții ale valorii *val* între pozițiile p_1 și p_2 ale tabloului A, unidimensional cu n elemente. Să se testeze funcția.

Observație. În program se va verifica corectitudinea argumentelor, dacă *val* nu poate fi calculată, un indicator de eroare va fi poziționat cu o valoare, iar dacă se poate calcula va primi o altă valoare.

De exemplu,

$$IER = \begin{cases} -2 & \text{pentru caz corect} \\ -1 & \text{pentru caz incorect} \end{cases}$$

27. Căutați un șir de caractere și afișați la imprimantă șirul în ordine inversă. Construiți o subrutină care să inverseze un șir de caractere.
28. Scrieți un program care să adune 2 numere romane.
29. Să se scrie o subrutină care să găsească valoarea maximă a elementelor unui tablou cu n elemente și poziția pe care se află primul element egal cu această valoare. Să se dea exemplu de folosire a subrutinei.
30. Scrieți un subprogram care să afișeze sub formă de text valoarea unui număr întreg pozitiv, cu cel mult trei cifre.

REZOLVARI

Paragraful 1.2

Rezolvarea problemei 1 a)

a	v s.	val. întregi
b	v s.	val întregi

citește a

citește b

serie b

serie a

stop

Rezolvarea problemei 2 a)

x	v s.	val. reale
f	v s.	val reale

citește x

atribuie $f \leftarrow \exp(-x) * \sin(x)$

serie f

stop

Rezolvarea problemei 2 b)

x	v s.	val. reale
y	v s.	val. reale
g	v s.	val. reale

citește x

citește y

atribuie $g \leftarrow \sqrt[3]{x^3 + y^3}$

serie g

stop

Rezolvarea problemei 2 c)

x	v s.	val reale
y	v s.	val reale
fi	v s.	val. reale
psi	v s.	val. reale

citește x

citește y

atribuie $fi \leftarrow x^2 * \cos y^2$

atribuie $psi \leftarrow y^2 + \ln(x^2 + 1)$

serie fi

serie psi

stop

Rezolvarea problemei 3

unghi	v.s.	val. reale
t	v.s.	val. reale
grade	v.s.	val. întregi
min	v.s.	val. întregi
sec	v.s.	val. întregi

citește unghi

```

atribuie t ← 180*unghi/3,14159
& grade ← [t]
& t ← (t - grade)*60
& min ← [t]
& t ← (t - min)*60
& sec ← [t]

```

serie grade, min, sec

stop

Observație. S-a notat cu [t] partea întreagă a lui t.

Paragraful 1.3. Rezolvări

Rezolvarea problemei 1 a)

m	v.s.	val. întregi
n	v.s.	val. întregi

```

citește  $m, n$ 
dacă  $m > n$  atunci
    serie  $m, n$ 
altfel
    serie  $n, m$ 
    ■
stop

```

Rezolvarea problemei 4

x	v.s.	val. reale
f	v.s.	val. reale

```

citește  $x$ 
dacă  $x < -1$  atunci
    atribuie  $f \leftarrow 1 - x$ 
    altfel
        dacă  $x \leq 1$  atunci
            atribuie  $f \leftarrow \sqrt{1 - x^2}$ 
            altfel
                atribuie  $f \leftarrow x - 1$ 
                ■
        ■
    serie  $f$ 
    stop

```

Rezolvarea problemei 5

ap	v.s.	v. întregi poz.
lp	v.s.	v. întregi poz.
zp	v.s.	v. întregi poz.
an	v.s.	v. întregi poz.
ln	v.s.	v. întregi poz.
zn	v.s.	v. întregi poz.
$virstă$	v.s.	v. întregi poz.

```

citește  $ap, lp, zp, an, ln, zn$ 
dacă  $ap < an$  atunci
    serie „date eronate”
    altfel
        atribuie  $virstă \leftarrow ap - an$ 
        dacă  $ln > lp \vee (ln = lp \wedge zn > zp)$  atunci
            atribuie  $virstă \leftarrow virstă - 1$ 
            ■
        ■
    serie  $virstă$ 
    stop

```

Rezolvarea problemei 7

a) Dacă notăm cu α_0 și β_0 valorile citite în m și n , iar cu α_1 și β_1 , α_2 și β_2 , ... valorile din m și n la sfârșitul execuțiilor succesive ale servenței din ciclu, avem relațiile

$$|\alpha_0 - \beta_0| > |\alpha_1 - \beta_1| > \dots > |\alpha_k - \beta_k| > \dots \geq 0$$

Deci,

$$(\exists n)(n \in \mathbb{N}) | \alpha_n - \beta_n | = 0 \Rightarrow \alpha_n = \beta_n$$

Această este relația care ne asigură ieșirea din ciclu

Știind că $\text{c.m.m.d.c.}(a, b) = \text{c.m.m.d.c.}(a - b, b)$ se poate arăta prin inducție că

$$\text{c.m.m.d.c.}(\alpha_k, \beta_k) = \text{c.m.m.d.c.}(\alpha_0, \beta_0) = \gamma.$$

Într-adevăr, pentru $i = 0$ egalitatea este evidentă

Să presupunem că: $\text{c.m.m.d.c.}(\alpha_j, \beta_j) = \gamma$ oricare ar fi $j \leq k$ și fie cazul $i = k + 1$. Vom avea

— fie $\alpha_{k+1} = \alpha_k - \beta_k$ și $\beta_{k+1} = \beta_k$

— fie $\alpha_{k+1} = \alpha_k$ și $\beta_{k+1} = \beta_k - \alpha_k$.

În ambele cazuri $\text{c.m.m.d.c.}(\alpha_{k+1}, \beta_{k+1}) = \gamma$

b)

m	v s	v. întregi strict poz.
n	v s	v. întregi strict poz.
$gata$	v s	v logice

citește m, n

atribuie $gata \leftarrow \text{fals}$

```

ciclul:
  dacă  $m = n$  atunci
    atribuie  $gata \leftarrow \text{adevărat}$ 
  altfel
    dacă  $m > n$  atunci
      atribuie  $m \leftarrow m - n$ 
    altfel
      atribuie  $n \leftarrow n - m$ 
  ■
pînă  $gata$ 
  scrie  $m$ 
stop
    
```

Rezolvarea problemei 8

n	v s	v. întregi poz.
i	v s	v. întregi poz.
f	v s	v. întregi poz.

citește n

atribuie $i \leftarrow 1$ & $f \leftarrow 1$

```

[ cît timp  $1 < i \leq n$  repetă
  atribuie  $f \leftarrow f * i$ 
  ■
]
    
```

scrie f

stop

Discuție

1° Factorialul este o funcție cu o creștere rapidă. Dacă $5! = 120$, în schimb $10! = 3628800$, iar $1000!$ este de ordinul lui 10^{2500} . De aceea calculele al căror număr de pași se exprimă cu ajutorul factorialului trebuie tratate cu cea mai mare atenție deoarece timpul lor de execuție poate crește rapid în funcție de datele de intrare.

2° Formula lui Stirling ne dă o alternativă simplă de calcul cu aproximație al factorialului

$$n! \approx \sqrt{2 \cdot \pi \cdot n} \left(\frac{n}{e} \right)^n$$

$$\text{Eroare relativă} \approx \text{circa } \frac{1}{12 \cdot n}.$$

Numele celulelor din memorie utilizate respectă notațiile din indicație

p	v.s.	val. întregi poz.
α	v.s.	val. reale
$xzero$	v.s.	val. reale
eps	v.s.	val. reale poz.
x	v.s.	val. reale
er	v.s.	val. reale

```

citește  $p, \alpha, xzero, eps$ 
atribuie  $x \leftarrow ((p - 1) * xzero + \alpha / xzero^{p-1}) / p$ 
dacă  $x \neq xzero$  atunci
  atribuie  $er \leftarrow |x|$ 
  cit timp  $er < eps$  repetă
    atribuie  $er \leftarrow er * (p - 1) * (1 - \alpha / x^p) / p$ 
    &  $x \leftarrow ((p - 1) * x + \alpha / x^{p-1}) / p$ 
scrie  $x$ 
stop
  
```

Rezolvarea problemei 11

a)

n	var. s	v. întregi poz.
v	tabl. 1 dim (1000)	v. reale
i	var. s	v. întregi poz.
max	var. s	v. reale

```

citește  $n$ 
citește  $(v(i), i = 1, n)$ 
atribuie  $max \leftarrow v(1) \& i \leftarrow 1$ 
cit timp  $i \leq n$  repetă
  dacă  $max < v(i)$  atunci
    atribuie  $max \leftarrow v(i)$ 
  atribuie  $i \leftarrow i + 1$ 
scrie  $max$ 
stop
  
```

b)

n	var. s	v. întregi poz.
v	var. s	v. reale
i	var. s	v. întregi poz.
max	var. s.	v. reale

```

citește  $n$ 
citește  $v$ 
atribuie  $max \leftarrow v \& i \leftarrow 1$ 
cit timp  $i \leq n$  repetă
  dacă  $max < v$  atunci
    atribuie  $max \leftarrow v$ 
  atribuie  $i \leftarrow i + 1$ 
scrie  $max$ 
stop
  
```

Observație Programul de la pct. b) necesită o zonă mică de date deoarece nu memorează tot șirul dintr-o dată ci citește de pe banda de intrare de îndată ce are nevoie de o nouă valoare. Soluția este posibilă, deoarece valorile citite sînt necesare o singură dată, după care ele se pot șterge. Situația diferă de cea întâlnită la algoritmul de ordonare.

Rezolvarea problemei 13

c)

<i>m</i>	var s	v. întregi poz.
<i>nr</i>	var s	v. reale
<i>prod</i>	var s	v. reale
<i>i</i>	var s	v. întregi poz.

```

citește m
atribuie prod ← 1 & i ← 1
cîlează
    citește nr
    dacă nr ≠ 0 atunci
        atribuie prod ← prod * nr
    ■
    atribuie i ← i + 1
pînă i > m
    ■
serie prod
stop
    
```

$h_1)$

<i>n</i>	var s	v. întregi poz.
<i>nr</i>	tab 1 d m	v. reale
<i>suma</i>	var s	v. reale
<i>i</i>	var s	v. întregi poz.
<i>medie</i>	var s	v. reale
<i>sumap</i>	var s	v. reale
<i>disp</i>	var s	v. reale
<i>abmp</i>	var s	v. reale

```

citește n
citește (nr(i), i = 1, n)
atribuie suma ← 0 & i ← 1
cît timp i ≤ n repetă
    atribuie suma ← suma + nr(i)
    & i ← i + 1
    ■
atribuie medie ← suma/n
atribuie sumap ← 0 & i ← 1
cît timp i ≤ n repetă
    atribuie sumap ← sumap + (nr(i) - medie)2
    & i ← i + 1
    ■
atribuie disp ← sumap/n
    & abmp ← √disp
    ■
serie disp, abmp
stop
    
```

$h_2)$

<i>n</i>	var s	v. întregi poz.
<i>nr</i>	var s	v. reale
<i>suma</i>	var s	v. reale
<i>i</i>	var s	v. întregi poz.
<i>sumap</i>	var s	v. întregi poz.
<i>disp</i>	var s	v. întregi poz.
<i>abmp</i>	var s	v. întregi poz.

```

citește n
citește nr
atribuie suma ← nr & sumap ← nr2
atribuie i ← 2
cît timp i ≤ n repetă
    citește nr
    atribuie suma ← suma + nr
    atribuie sumap ← sumap + nr2
    atribuie i ← i + 1
    ■
atribuie disp ← sumap/n - (suma/n)2
atribuie abmp ← √disp
serie disp, abmp
stop
    
```

Observație Comparând cei doi algoritmi de la pct. h (h_1 și h_2) constatăm că h_2 folosește mai eficient memoria și timpul de execuție, deoarece valorile nu se memorează toate deodată, prelucrarea realizându-se valoare cu valoare. Acest lucru a fost posibil printr-o transformare a relației ce dă dispersia astfel:

$$D = \frac{\sum_{i=1}^n (x_i - m)^2}{n} = \frac{\sum_{i=1}^n (x_i^2 - 2x_i \cdot m + m^2)}{n} = \frac{\sum_{i=1}^n x_i^2}{n} - 2 \cdot \frac{\sum_{i=1}^n x_i}{n} \cdot m + \frac{\sum_{i=1}^n m^2}{n} = \frac{\sum_{i=1}^n x_i^2}{n} - 2m^2 + m^2 = \frac{\sum_{i=1}^n x_i^2}{n} - \left(\frac{\sum_{i=1}^n x_i}{n} \right)^2$$

Rezolvarea problemei 14 b

În afara notațiilor făcute pentru programele din figura 1.27 mai utilizăm, k — nume al celulei care memorează rangul ultimei valori ce a trebuit să fie modificată în timpul căutării — se inițializează cu 0 și constituie astfel și un „semafor“ pentru ieșirea din ciclu, lim — nume al celulei ce memorează între diferitele parcurgeri ale șirului rangul ultimei inversări

n	var. simplă	v. întregi [1, 100]
a	tab. 1 dim (100)	v. reale
i	var. simplă	v. întregi
l	var. simplă	v. reale
k	var. simplă	v. întregi poz.
lim	var. simplă	v. întregi poz.
m	var. simplă	v. întregi

```

citește  $n$ 
citește  $(a(i), i = 1, n)$ 
atribuie  $lim \leftarrow n$ 
ciclează
  atribuie  $k \leftarrow 0$  &  $m \leftarrow lim - 1$ 
  pentru  $i = 1, m$  execută
    dacă  $a(i) < a(i + 1)$  atunci
      atribuie  $l \leftarrow a(i)$  &  $a(i) \leftarrow a(i + 1)$  &  $a(i + 1) \leftarrow l$ 
      atribuie  $k \leftarrow i$ 
  atribuie  $lim \leftarrow k$ 
  până  $k = 0$ 
serie  $(a(i), i = 1, n)$ 
stop
  
```

Rezolvarea problemei 14 c

Folosim majoritatea notațiilor algoritmului din figura 1.27. Celelalte rezultă imediat din context

```

citește  $n$ 
citește  $(a(i), i = 1, n)$ 
atribuie  $m \leftarrow n - 1$ 
pentru  $i = 1, m$  execută
  atribuie  $p \leftarrow i + 1$ 
  pentru  $j = p, n$  execută
    dacă  $a(i) < a(j)$  atunci
      atribuie  $l \leftarrow a(i)$  &  $a(i) \leftarrow a(j)$  &  $a(j) \leftarrow l$ 
serie  $(a(i), i = 1, n)$ 
stop
  
```

Rezolvarea problemei 16

n	var. s.	v. întregi poz.
a	tabl., 2 dim. (20, 20)	v. reale
b	tabl., 1 dim. (20)	v. reale
x	tabl., 1 dim. (20)	v. reale
i	var. s.	v. întregi
j	var s	v. întregi
k	var s	v. întregi
s	var s	v. reale

citește n

citește $\{(a(i, j), j = 1, n), i = 1, n\}, \{b(i), i = 1, n\}$

atribuie $x(1) \leftarrow b(1)/a(1, 1)$

```

pentru  $i = 2, n$  execută
    atribuie  $s \leftarrow b(i)$  &  $k \leftarrow i - 1$ 
    pentru  $j = 1, k$  execută
        atribuie  $s \leftarrow s - a(i, j) \cdot x(j)$ 
    atribuie  $x(i) \leftarrow s/a(i, i)$ 

```

scrie $(x(i), i = 1, n)$

stop

Rezolvarea problemei 19

n	var s.	v. întregi poz.
$număr$	var. s	v. reale
nr	tabl., 2 dim. (1000, 2)	v. reale
k	var. s	v. întregi poz.
j	var s.	v. întregi poz.
i	var s.	v. întregi poz.
$găsit$	var s.	v. logice

2. a) IF(A LT 0 AND B.GT 0,OR ALFA.EQ 0) X = - 2
 b) || IF(X.GT.0.000001.AND.X.LE.1) GO TO 777
 STOP
3. || IF(K - K/2,2.EQ.0) GO TO 615
5. a) 28 00 b) 11.8
7. || INTEGER S
 READ (105, 10) N
 S = 0
 K = 1
 1 S = S + K
 K = K + 1
 IF(K LE N) GO TO 1
 WRITE (108, 11) S
 STOP
 10 FORMAT (I3)
 11 FORMAT ('□', 17)
 END

Paragraful II.3.3

1. Permiso: 2, 3, 6, 8, 9, 10, 11, 16, 18, 19; Interzis: 1, 4, 5, 7, 12, 13, 14, 15, 17, 20

2. a

3. Corecto: a, b, d, e, g.

6. || INTEGER A (20)
 A (1) = 1
 A (2) = 1
 DO 1 K = 3, 20
 1 A(K) = A(K - 2) + A(K - 1)
 WRITE (108, 11) A
 STOP
 11 FORMAT ('□', 1019)
 END
11. || INTEGER FACT
 N = . . .
 FACT = 1
 DO 2 K = 1, N
 FACT = FACT * K
 WRITE (108, 11) FACT
 IF (FACT. GT.N) GO TO 3
 2 CONTINUE
 3 V = 9999

12. || REAL A (30), B(30), X(30)

 DO 1 I = 1, 30
 IF (A(I). LT. B(I)) X(I) = - 1
 IF (A(I).EQ B (I)) X(I) = 0
 1 IF (A(I).GT.B(I)) X(I) = 1

6.

20.

- 1 || REAL FUNCTION PVECT (A, K)
 REAL A (N).
 PVECT = 1
 DO 1 I = 1, K
 PVECT = PVECT * A(I)
 RETURN
 END
- 1 + || FUNCTION PR0D (X, N, M)
 REAL X (M, N)
 PR0D = 1
 DO 1 I = 1, M
 DO 1 J = 1, N
 PR0D = PR0D * AM0D X (1, J)
 **2
 RETURN
 END

CUPRINS

Cap. I. ELEMENTE DE BAZĂ ALE PROGRAMĂRII CALCULATORILOR	3
I.1. Sistemul de prelucrare a datelor simplu (SPDS)	4
I.1.1. Structura și funcționarea SPDS	4
I.1.2. Funcționarea calculatorului C	7
I.1.3. Starea calculatorului C	10
I.1.4. Programarea calculatorului C	12
I.2. Reprezentarea algoritmilor în limbajul LPS	14
I.2.1. Definirea limbajelor de programare	14
I.2.2. Instrucțiunile de bază ale limbajului LPS	15
I.2.3. Secvența de instrucțiuni	18
I.2.4. Reprezentarea algoritmilor în LPS	21
I.2.5. Probleme	24
I.3. Instrucțiunile de control ale limbajului LPS	25
I.3.1. Instrucțiunea condițională	25
I.3.2. Instrucțiuni de ciclare cu condiție	29
I.3.3. Utilizarea tablourilor în LPS	34
I.3.4. Instrucțiunea de ciclare cu contor	41
I.3.5. Probleme	43
Cap. II. PROGRAMAREA ÎN LIMBAJUL FORTRAN	50
II.1. Date FORTRAN. Instrucțiunea de atribuire. Expresii	50
II.1.1. Introducere	50
II.1.2. Etapele rezolvării unei probleme utilizând limbajul FORTRAN	51
II.1.3. Date FORTRAN	56
II.1.4. Declarații FORTRAN	59
II.1.5. Instrucțiuni de atribuire	60
II.1.6. Expresii FORTRAN	61
II.1.7. Exerciții	66
II.2. Principalele instrucțiuni ale limbajului FORTRAN	69
II.2.1. Introducere	69
II.2.2. Instrucțiunea condițională	70
II.2.2.1. Instrucțiunea IF logic	70
II.2.2.2. Instrucțiunea IF aritmetic	70
II.2.3. Instrucțiuni de salt necondiționat	70
II.2.4. Instrucțiunile: CONTINUE, STOP, PAUSE și END	71
II.2.5. Instrucțiuni de intrare/ieșire	72

II.2.5.1. Instrucțiunea de citire	73
II.2.5.2. Instrucțiunea de scriere	74
II.2.6. Instrucțiunea FORMAT	74
II.2.6.1. Descriptori de reprezentare a valorilor variabilelor	75
II.2.6.2. Așezarea cimpurilor în înregistrare și a înregistrărilor în cadrul setului de date	77
II.2.7. Reguli de alcătuire a listei FORMAT	79
II.2.8. Reguli după care se explorează lista FORMAT	79
II.2.9. Exemple de programe FORTRAN	81
II.2.10. Exerciții	84
II.3. Facilități de programare în limbajul FORTRAN	85
II.3.1. Instrucțiuni DØ	85
II.3.2. Subprograme	87
II.3.2.1. Funcții externe	88
II.3.2.2. Subrutine	91
II.3.3. Exerciții	93
Rezolvări	96

